

# **TPM Main Part 3 Commands**

**Specification Version 1.2  
Level 2 Revision 94  
29 March 2006**

Contact: [tpmwg@trustedcomputinggroup.org](mailto:tpmwg@trustedcomputinggroup.org)

## **TCG Published**

Copyright © TCG 2003-2006

**TCG**

Copyright © 2003-2006 Trusted Computing Group, Incorporated.

## **Disclaimer**

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. Without limitation, TCG disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification and to the implementation of this specification, and TCG disclaims all liability for cost of procurement of substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, direct, indirect, or special damages, whether under contract, tort, warranty or otherwise, arising in any way out of use or reliance upon this specification or any information herein.

No license, express or implied, by estoppel or otherwise, to any TCG or TCG member intellectual property rights is granted herein.

**Except that a license is hereby granted by TCG to copy and reproduce this specification for internal use only.**

Contact the Trusted Computing Group at [www.trustedcomputinggroup.org](http://www.trustedcomputinggroup.org) for information on specification licensing through membership agreements.

Any marks and brands contained herein are the property of their respective owners.

## **Acknowledgement**

TCG wishes to thank all those who contributed to this specification. This version builds on the work published in version 1.1 and those who helped on that version have helped on this version.

A special thank you goes to the members of the TPM workgroup who had early access to this version and made invaluable contributions, corrections and support.

David Grawrock

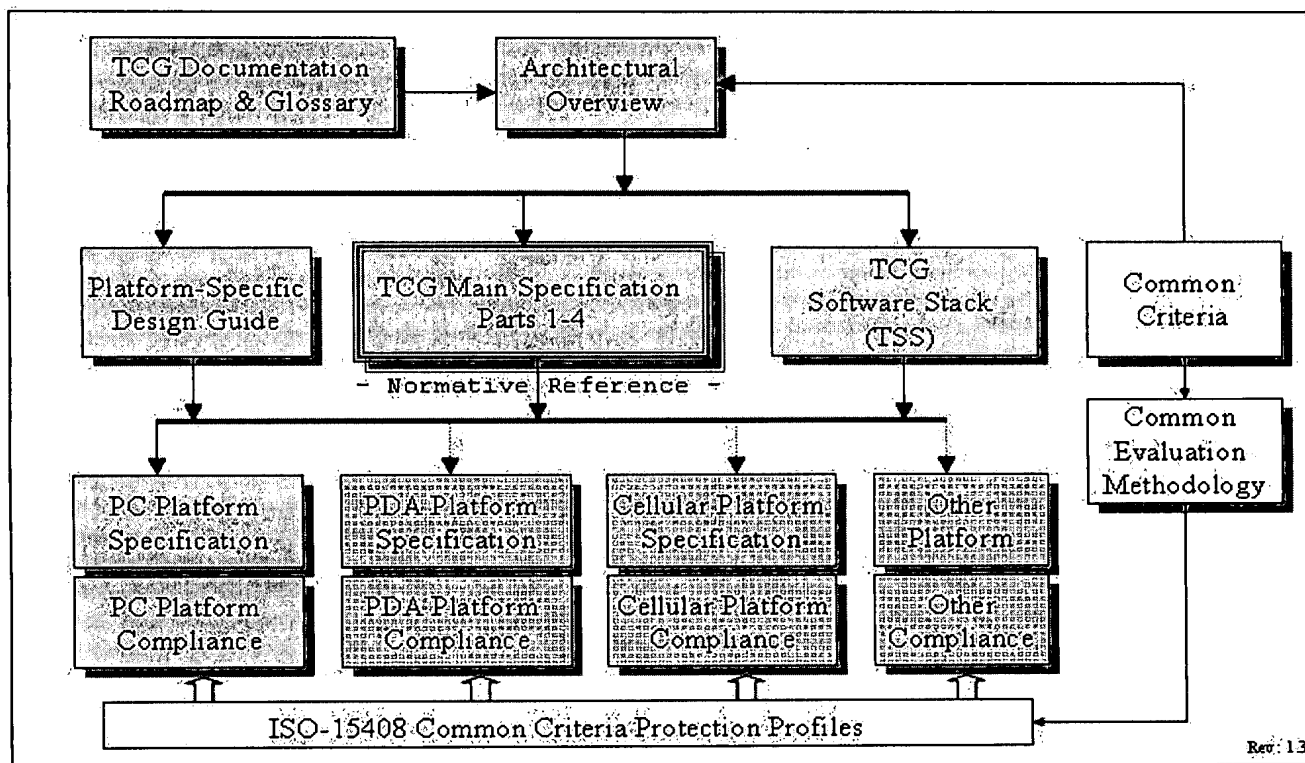
TPM Workgroup chair

**Change History**

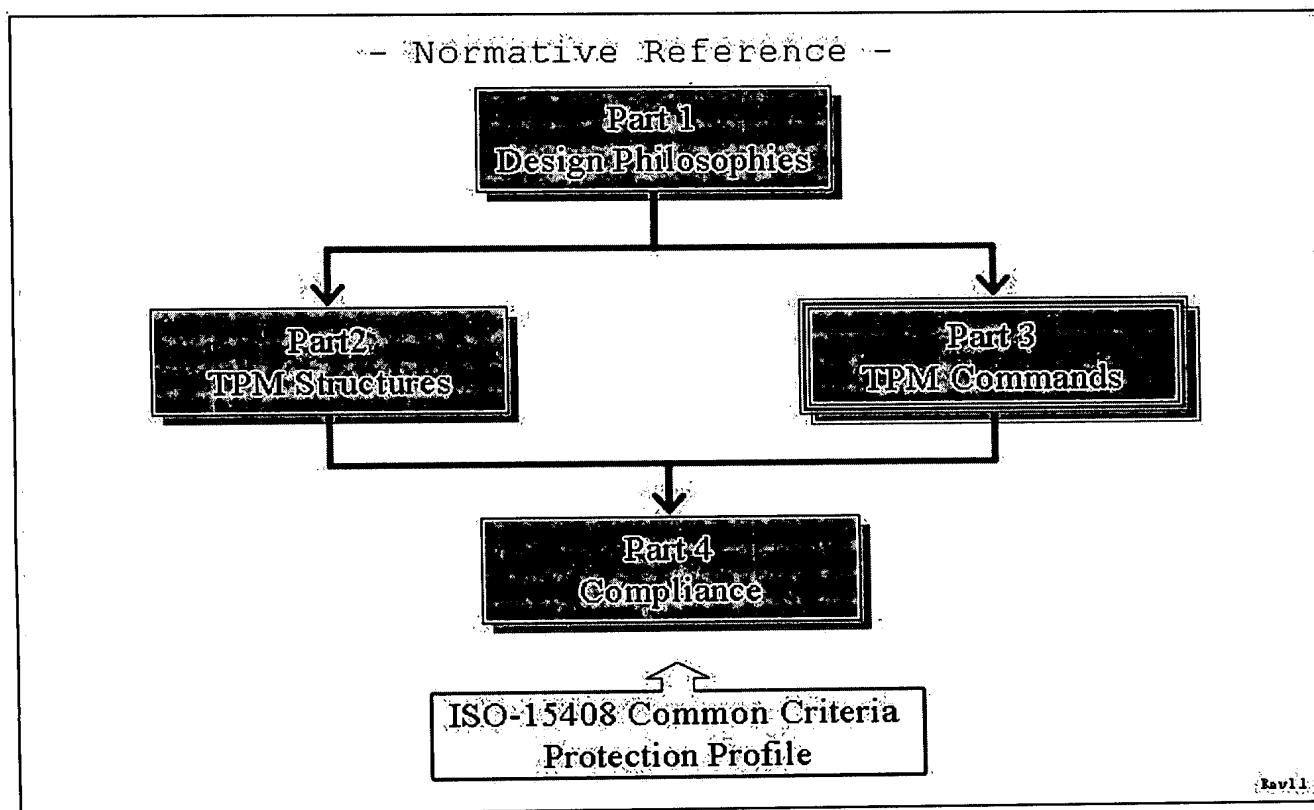
Version	Date	Description
Rev 50	Jul 2003	Started 01 Jul 2003 by David Grawrock Breakup into parts and the merge of 1.1 commands
Rev 63	Oct 2003	Change history tied to part 1 and kept in part 1 (DP)
Rev 71	Mar 2004	Change in terms from authorization data to AuthData.
Rev 91	Sept 2005	The following modifications were made by Tasneem Brutch: <ul style="list-style-type: none"> <li>▪ Update to section 6.2 informative, for TPM_OwnerClear.</li> <li>▪ Addition of action item 15, to section 6.2, for TPM_OwnerClear.</li> <li>▪ Addition of "MAY" to section 20.1, TPM_NV_DefineSpace, Action 1(a).</li> <li>▪ Addition of a new Action (4) to Section 20.2, TPM_NV_WriteValue</li> <li>▪ Addition of a new Action (3) to Section 20.4, TPM_NV_ReadValue.</li> <li>▪ Typo corrected in Section 21.1</li> <li>▪ Moved TPM_GetCapabilityOwner from Section the Deleted Commands (section 28.1) to section 7.3. Added information on operands, command description and actions from Rev. 67.</li> </ul>
Rev 92	Sept 2005	Section 7.3 TPM_GetCapabilityOwner Ordinal was added to the outgoing params, which is not returned but is typically included in outParamDigest.
Rev 92	Sept 2005	Corrected a copy and paste error. Part 3 20.2 TPM_NV_WriteValue Removed the Action "3. If D1 -> TPM_NV_PER_AUTHREAD is TRUE return TPM_AUTH_CONFLICT"
Rev 93	Sept. 2005	Moved TPM_CertifySelfTest command to the deleted section.



# TCG Doc Roadmap – Main Spec



# TCG Main Spec Roadmap



## Table of Contents

1. Scope and Audience.....	1
1.1 Key words .....	2
1.2 Statement Type .....	3
2. Description and TODO.....	4
3. Admin Startup and State .....	5
3.1 TPM_Init .....	5
3.2 TPM_Startup.....	6
3.3 TPM_SaveState .....	9
4. Admin Testing .....	11
4.1 TPM_SelfTestFull .....	11
4.2 TPM_ContinueSelfTest .....	12
4.3 TPM_GetTestResult .....	14
5. Admin Opt-in .....	15
5.1 TPM_SetOwnerInstall .....	15
5.2 TPM_OwnerSetDisable .....	16
5.3 TPM_PhysicalEnable.....	17
5.4 TPM_PhysicalDisable .....	18
5.5 TPM_PhysicalSetDeactivated.....	19
5.6 TPM_SetTempDeactivated.....	20
5.7 TPM_SetOperatorAuth.....	22
6. Admin Ownership .....	23
6.1 TPM_TakeOwnership .....	23
6.2 TPM_OwnerClear.....	26
6.3 TPM_ForceClear .....	29
6.4 TPM_DisableOwnerClear .....	30
6.5 TPM_DisableForceClear .....	32
6.6 TSC_PhysicalPresence.....	33
6.7 TSC_ResetEstablishmentBit.....	36
7. The Capability Commands .....	37
7.1 TPM_GetCapability .....	38
7.2 TPM_SetCapability .....	39
7.3 TPM_GetCapabilityOwner .....	41
8. Auditing .....	43
8.1 Audit Generation.....	43
8.2 Effect of audit failing after completion of a command .....	45

8.3	TPM_GetAuditDigest .....	46
8.4	TPM_GetAuditDigestSigned .....	48
8.5	TPM_SetOrdinalAuditStatus .....	51
9.	Administrative Functions - Management .....	52
9.1	TPM_FieldUpgrade .....	52
9.2	TPM_SetRedirection .....	54
9.3	TPM_ResetLockValue .....	56
10.	Storage functions .....	58
10.1	TPM_Seal .....	58
10.2	TPM_Unseal .....	62
10.3	TPM_UnBind .....	66
10.4	TPM_CreateWrapKey .....	69
10.5	TPM_LoadKey2 .....	72
10.6	TPM_GetPubKey .....	76
10.7	TPM_Sealx .....	78
11.	Migration .....	82
11.1	TPM_CreateMigrationBlob .....	82
11.2	TPM_ConvertMigrationBlob .....	86
11.3	TPM_AuthorizeMigrationKey .....	88
11.4	TPM_MigrateKey .....	90
11.5	TPM_CMK_SetRestrictions .....	92
11.6	TPM_CMK_ApproveMA .....	94
11.7	TPM_CMK_CreateKey .....	96
11.8	TPM_CMK_CreateTicket .....	99
11.9	TPM_CMK_CreateBlob .....	101
11.10	TPM_CMK_ConvertMigration .....	106
12.	Maintenance Functions (optional) .....	109
12.1	TPM_CreateMaintenanceArchive .....	110
12.2	TPM_LoadMaintenanceArchive .....	112
12.3	TPM_KillMaintenanceFeature .....	115
12.4	TPM_LoadManuMaintPub .....	116
12.5	TPM_ReadManuMaintPub .....	118
13.	Cryptographic Functions .....	119
13.1	TPM_SHA1Start .....	119
13.2	TPM_SHA1Update .....	120
13.3	TPM_SHA1Complete .....	121
13.4	TPM_SHA1CompleteExtend .....	122

13.5	TPM_Sign .....	124
13.6	TPM_GetRandom.....	126
13.7	TPM_StirRandom .....	127
13.8	TPM_CertifyKey .....	128
13.9	TPM_CertifyKey2 .....	133
14.	Endorsement Key Handling .....	137
14.1	TPM_CreateEndorsementKeyPair .....	138
14.2	TPM_CreateRevocableEK.....	140
14.3	TPM_RevokeTrust.....	142
14.4	TPM_ReadPubek .....	143
14.5	TPM_OwnerReadInternalPub .....	144
15.	Identity Creation and Activation .....	146
15.1	TPM_MakeIdentity .....	146
15.2	TPM_ActivateIdentity .....	150
16.	Integrity Collection and Reporting .....	153
16.1	TPM_Extend .....	154
16.2	TPM_PCRRead.....	156
16.3	TPM_Quote.....	157
16.4	TPM_PCR_Reset .....	159
16.5	TPM_Quote2 .....	161
17.	Changing AuthData .....	164
17.1	TPM_ChangeAuth .....	164
17.2	TPM_ChangeAuthOwner .....	167
18.	Authorization Sessions .....	169
18.1	TPM_OIAP.....	169
18.1.1	Actions to validate an OIAP session .....	169
18.2	TPM_OSAF .....	172
18.2.1	Actions to validate an OSAF session .....	174
18.3	TPM_DSAP .....	176
18.4	TPM_SetOwnerPointer .....	180
19.	Delegation Commands .....	182
19.1	TPM_Delegate_Manage .....	182
19.2	TPM_Delegate_CreateKeyDelegation .....	186
19.3	TPM_Delegate_CreateOwnerDelegation.....	189
19.4	TPM_Delegate_LoadOwnerDelegation .....	192
19.5	TPM_Delegate_ReadTable .....	195
19.6	TPM_Delegate_UpdateVerification .....	197

19.7	TPM_Delegate_VerifyDelegation .....	200
20.	Non-volatile Storage .....	202
20.1	TPM_NV_DefineSpace .....	203
20.2	TPM_NV_WriteValue .....	207
20.3	TPM_NV_WriteValueAuth .....	210
20.4	TPM_NV_ReadValue .....	212
20.5	TPM_NV_ReadValueAuth .....	214
21.	Session Management .....	216
21.1	TPM_KeyControlOwner .....	216
21.2	TPM_SaveContext .....	219
21.3	TPM_LoadContext .....	222
22.	Eviction .....	224
22.1	TPM_FlushSpecific .....	225
23.	Timing Ticks .....	227
23.1	TPM_GetTicks .....	227
23.2	TPM_TickStampBlob .....	228
24.	Transport Sessions .....	231
24.1	TPM_EstablishTransport .....	231
24.2	TPM_ExecuteTransport .....	235
24.3	TPM_ReleaseTransportSigned .....	242
25.	Monotonic Counter .....	245
25.1	TPM_CreateCounter .....	245
25.2	TPM_IncrementCounter .....	248
25.3	TPM_ReadCounter .....	250
25.4	TPM_ReleaseCounter .....	251
25.5	TPM_ReleaseCounterOwner .....	253
26.	DAA commands .....	255
26.1	TPM_DAA_Join .....	255
26.2	TPM_DAA_Sign .....	272
27.	Deprecated commands .....	283
27.1	Key commands .....	284
27.1.1	TPM_EvictKey .....	284
27.1.2	TPM_Terminate_Handle .....	285
27.2	Context management .....	287
27.2.1	TPM_SaveKeyContext .....	287
27.2.2	TPM_LoadKeyContext .....	289
27.2.3	TPM_SaveAuthContext .....	290

27.2.4	TPM_LoadAuthContext .....	291
27.3	DIR commands .....	292
27.3.1	TPM_DirWriteAuth .....	293
27.3.2	TPM_DirRead .....	295
27.4	Change Auth .....	296
27.4.1	TPM_ChangeAuthAsymStart .....	297
27.4.2	TPM_ChangeAuthAsymFinish .....	300
27.5	TPM_Reset .....	303
27.6	TPM_OwnerReadPubek .....	304
27.7	TPM_DisablePubekRead .....	305
27.8	TPM_LoadKey .....	306
28.	Deleted Commands .....	310
28.1	TPM_GetCapabilitySigned .....	311
28.2	TPM_GetOrdinalAuditStatus .....	312
28.3	TPM_CertifySelfTest .....	313

End of Introduction do not delete

## **1. Scope and Audience**

The TPCA main specification is an industry specification that enables trust in computing platforms in general. The main specification is broken into parts to make the role of each document clear. A version of the specification (like 1.2) requires all parts to be a complete specification.

This is Part 3 the structures that the TPM will use.

This document is an industry specification that enables trust in computing platforms in general.



## 9    **1.1        Key words**

10    The key words “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,”  
11    “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” in the chapters 2-10  
12    normative statements are to be interpreted as described in [RFC-2119].

## 1.2 Statement Type

Please note a very important distinction between different sections of text throughout this document. You will encounter two distinctive kinds of text: informative comment and normative statements. Because most of the text in this specification will be of the kind normative statements, the authors have informally defined it as the default and, as such, have specifically called out text of the kind informative comment. They have done this by flagging the beginning and end of each informative comment and highlighting its text in gray. This means that unless text is specifically marked as of the kind informative comment, you can consider it of the kind normative statements.

For example:

**Start of informative comment:**

This is the first paragraph of 1-n paragraphs containing text of the kind informative comment ...

This is the second paragraph of text of the kind informative comment ...

This is the nth paragraph of text of the kind informative comment ...

To understand the TPM specification the user must read the specification. (This use of MUST does not require any action).

**End of informative comment.**

This is the first paragraph of one or more paragraphs (and/or sections) containing the text of the kind normative statements ...

To understand the TPM specification the user MUST read the specification. (This use of MUST indicates a keyword usage and requires an action).

## **2. Description and TODO**

This document is to show the changes necessary to create the 1.2 version of the TCG specification. Some of the sections are brand new text; some are rewritten sections of the 1.1 version. Upon approval of the 1.2 changes, there will be a merging of the 1.1 and 1.2 versions to create a single 1.2 document.

## 3. Admin Startup and State

### Start of informative comment:

This section is the commands that start a TPM.

### End of informative comment.

### 3.1 TPM\_Init

#### Start of informative comment:

TPM\_Init is a physical method of initializing a TPM. There is no TPM\_Init ordinal as this is a platform message sent on the platform internals to the TPM. On a PC this command arrives at the TPM via the LPC bus and informs the TPM that the platform is performing a boot process.

TPM\_Init puts the TPM into a state where it waits for the command TPM\_Startup (which specifies the type of initialization that is required).

#### End of informative comment.

#### Definition

TPM\_Init();

Operation of the TPM. This is not a command that any software can execute. It is inherent in the design of the TPM and the platform that the TPM resides on.

#### Parameters

None

#### Description

1. The TPM\_Init signal indicates to the TPM that platform initialization is taking place. The TPM SHALL set the TPM into a state such that the only legal command to receive after the TPM\_Init is the TPM\_Startup command. The TPM\_Startup will further indicate to the TPM how to handle and initialize the TPM resources.
2. The platform design MUST be that the TPM is not the only component undergoing initialization. If the TPM\_Init signal forces the TPM to perform initialization then the platform MUST ensure that ALL components of the platform receive an initialization signal. This is to prevent an attacker from causing the TPM to initialize to a state where various masquerades are allowable. For instance, on a PC causing the TPM to initialize and expect measurements in PCR0 but the remainder of the platform does not initialize.
3. The design of the TPM MUST be such that the ONLY mechanism that signals TPM\_Init also signals initialization to the other platform components.

#### Actions

1. The TPM sets TPM\_STANY\_FLAGS -> postInitialise to TRUE.

## 3.2 TPM\_Startup

### Start of informative comment:

TPM\_Startup is always preceded by TPM\_Init, which is the physical indication (a system-wide reset) that TPM initialization is necessary.

There are many events on a platform that can cause a reset and the response to these events can require different operations to occur on the TPM. The mere reset indication does not contain sufficient information to inform the TPM as to what type of reset is occurring. Additional information known by the platform initialization code needs transmitting to the TPM. The TPM\_Startup command provides the mechanism to transmit the information.

The TPM can startup in three different modes:

A "clear" start where all variables go back to their default or non-volatile set state

A "save" start where the TPM recovers appropriate information and restores various values based on a prior TPM\_SaveState. This recovery requires an invocation of TPM\_Init to be successful.

A failing "save" start must shut down the TPM. The CRTM cannot leave the TPM in a state where an untrusted upper software layer could issue a "clear" and then extend PCR's and thus mimic the CRTM.

A "deactivated" start where the TPM turns itself off and requires another TPM\_Init before the TPM will execute in a fully operational state.

### End of informative comment.

### Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_Startup
4	2	2S	2	TPM_STARTUP_TYPE	startupType	Type of startup that is occurring

### Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Startup

### Description

TPM\_Startup MUST be generated by a trusted entity (the RTM or the TPM, for example).

## 99 Actions

1. If TPM\_STANY\_FLAGS -> postInitialise is FALSE,
  - a. Then the TPM MUST return TPM\_INVALID\_POSTINIT, and exit this capability
2. If stType = TPM\_ST\_CLEAR
  - a. Ensure that sessions associated with resources TPM\_RT\_CONTEXT, TPM\_RT\_AUTH and TPM\_RT\_TRANS are invalidated
  - b. Reset TPM\_STCLEAR\_DATA -> PCR[] values to each correct default value
    - i. pcrReset is FALSE, set to 0x00..00
    - ii. pcrReset is TRUE, set to 0xFF..FF
  - c. Set the following TPM\_STCLEAR\_FLAGS to their default state
    - i. PhysicalPresence
    - ii. PhysicalPresenceLock
    - iii. disableForceClear
  - d. The TPM MAY initialize auditDigest to NULL
    - i. If not initialized to NULL the TPM SHALL ensure that auditDigest contains a valid value
    - ii. If initialization fails the TPM SHALL set auditDigest to NULL and SHALL set the internal TPM state so that the TPM returns TPM\_FAILEDSELFTEST to all subsequent commands.
  - e. The TPM SHALL set TPM\_STCLEAR\_FLAGS -> deactivated to the same state as TPM\_PERMANENT\_FLAGS -> deactivated
  - f. The TPM MUST set the TPM\_STANY\_DATA fields to:
    - i. TPM\_STANY\_DATA->contextNonceSession is set to NULLS
    - ii. TPM\_STANY\_DATA->contextCount is set to 0
    - iii. TPM\_STANY\_DATA->contextList is set to 0
  - g. The TPM MUST set TPM\_STCLEAR\_DATA fields to:
    - i. Invalidate contextNonceKey
    - ii. countID to NULL
    - iii. ownerReference to TPM\_KH\_OWNER
  - h. The TPM MUST set the following TPM\_STCLEAR\_FLAGS to
    - i. bGlobalLock to FALSE
  - i. Determine which keys should remain in the TPM
    - i. For each key that has a valid preserved value in the TPM
      - (1) if parentPCRStatus is TRUE then call TPM\_FlushSpecific(keyHandle)
      - (2) if IsVolatile is TRUE then call TPM\_FlushSpecific(keyHandle)
    - ii. Keys under control of the OwnerEvict flag MUST stay resident in the TPM

- 135 3. If stType = TPM\_ST\_STATE
- 136 a. If the TPM has no state to restore the TPM MUST set the internal state such that it
- 137 returns TPM\_FAILEDSELFTEST to all subsequent commands
- 138 b. The TPM MAY determine for each session type (authorization, transport...) to release
- 139 or maintain the session information. The TPM reports how it manages sessions in the
- 140 TPM\_GetCapability command.
- 141 c. The TPM SHALL take all necessary actions to ensure that all PCRs contain valid
- 142 preserved values. If the TPM is unable to successfully complete these actions, it SHALL
- 143 enter the TPM failure mode.
- 144 i. For resettable PCR the TPM MUST set the value of TPM\_STCLEAR\_DATA ->
- 145 PCR[] to the resettable PCR default value. The TPM MUST NOT restore a resettable
- 146 PCR to a preserved value
- 147 d. The TPM MAY initialize auditDigest to NULL
- 148 i. Otherwise, the TPM SHALL take all actions necessary to ensure that auditDigest
- 149 contains a valid value. If the TPM is unable to successfully complete these
- 150 actions, the TPM SHALL initialize auditDigest to NULL and SHALL set the internal
- 151 set such that the TPM returns TPM\_FAILEDSELFTEST to all subsequent
- 152 commands.
- 153 e. The TPM MUST restore the following flags to their preserved states:
- 154 i. All values in TPM\_STCLEAR\_FLAGS
- 155 ii. All values in TPM\_STCLEAR\_DATA
- 156 f. The TPM MUST restore all keys that have a valid preserved value
- 157 g. The TPM resumes normal operation. If the TPM is unable to resume normal
- 158 operation, it SHALL enter the TPM failure mode.
- 159 4. If stType = TPM\_ST\_DEACTIVATED
- 160 a. Invalidate sessions
- 161 i. Ensure that all resources associated with saved and active sessions are
- 162 invalidated
- 163 b. Set the TPM\_STCLEAR\_FLAGS to their default state.
- 164 c. Set TPM\_STCLEAR\_FLAGS -> deactivated to TRUE
- 165 5. The TPM MUST ensure that state associated with TPM\_SaveState is invalidated
- 166 6. The TPM MUST set TPM\_STANY\_FLAGS -> postInitialise to FALSE
- 167 a. postInitialize is set to FALSE even if the TPM is in failure mode.

### 3.3 TPM\_SaveState

#### Start of informative comment:

This warns a TPM to save some state information.

If the relevant shielded storage is non-volatile, this command need have no effect.

If the relevant shielded storage is volatile and the TPM alone is unable to detect the loss of external power in time to move data to non-volatile memory, this command should be presented before the TPM enters a low or no power state.

#### End of informative comment.

#### Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveState.

#### Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveState.

#### Description

1. Preserved values MUST be non-volatile.
2. If data is never stored in a volatile medium, that data MAY be used as preserved data. In such cases, no explicit action may be required to preserve that data.
3. If an explicit action is required to preserve data, it MUST be possible for the TPM to determine whether preserved data is valid.
4. If a parameter mirrored by any preserved value is altered, all preserved values MUST be declared invalid.
5. The TPM MAY declare all preserved values invalid in response to any command other than TPM\_Init.

#### Actions

1. Store TPM\_STCLEAR\_DATA -> PCR contents except for
  - a. If the PCR attribute pcrReset is TRUE
  - b. Any platform identified debug PCR



- 192 2. The auditDigest MUST be handled according to the audit requirements as reported by  
193 TPM\_GetCapability
- 194 a. If the ordinalAuditStatus is TRUE for the TPM\_SaveState ordinal and the auditDigest  
195 is being stored in the saved state, the saved auditDigest MUST include the  
196 TPM\_SaveState input parameters and MUST NOT include the output parameters.
- 197 3. All values in TPM\_STCLEAR\_DATA MUST be preserved
- 198 4. All values in TPM\_STCLEAR\_FLAGS MUST be preserved
- 199 5. The contents of any key that is currently loaded SHOULD be preserved if the key's  
200 parentPCRStatus indicator is FALSE and its IsVolatile indicator is FALSE.
- 201 6. The contents of any key that has TPM\_KEY\_CONTROL\_OWNER\_EVICT set MUST be  
202 preserved
- 203 7. The contents of any key that is currently loaded MAY be preserved as reported by  
204 TPM\_GetCapability
- 205 8. The contents of sessions (authorization, transport etc.) MAY be preserved as reported by  
206 TPM\_GetCapability

## 4. Admin Testing

### 4.1 TPM\_SelfTestFull

**Start of informative comment:**

TPM\_SelfTestFull tests all of the TPM capabilities.

Unlike TPM\_ContinueSelfTest, which may optionally return immediately and then perform the tests, TPM\_SelfTestFull always performs the tests and then returns success or failure.

**End of informative comment.**

#### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SelfTestFull

#### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SelfTestFull

#### Actions

1. TPM\_SelfTestFull SHALL cause a TPM to perform self-test of each TPM internal function.
  - a. If the self-test succeeds, return TPM\_SUCCESS.
  - b. If the self-test fails, return TPM\_FAILEDSELFTEST.
2. Failure of any test results in overall failure, and the TPM goes into failure mode.
3. If the TPM has not executed the action of TPM\_ContinueSelfTest, the TPM
  - a. MAY perform the full self-test.
  - b. MAY return TPM\_NEEDS\_SELFTEST.

## 4.2 TPM\_ContinueSelfTest

### Start of informative comment:

TPM\_ContinueSelfTest informs the TPM that it should complete the self-test of all TPM functions.

The TPM may return success immediately and then perform the self-test, or it may perform the self-test and then return success or failure.

### End of informative comment.

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ContinueSelfTest

### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ContinueSelfTest

### Description

1. Prior to executing the actions of TPM\_ContinueSelfTest, if the TPM receives a command C1 that uses an untested TPM function, the TPM MUST take one of these actions:
  - a. The TPM MAY return TPM\_NEEDS\_SELFTEST
    - i. This indicates that the TPM has not tested the internal resources required to execute C1.
    - ii. The TPM does not execute C1.
    - iii. The caller MUST issue TPM\_ContinueSelfTest before re-issuing the command C1.
      - (1) If the TPM permits TPM\_SelfTestFull prior to completing the actions of TPM\_ContinueSelfTest, the caller MAY issue TPM\_SelfTestFull rather than TPM\_ContinueSelfTest.
  - b. The TPM MAY return TPM\_DOING\_SELFTEST
    - i. This indicates that the TPM is doing the actions of TPM\_ContinueSelfTest implicitly, as if the TPM\_ContinueSelfTest command had been issued.
    - ii. The TPM does not execute C1.

248           iii. The caller MUST wait for the actions of TPM\_ContinueSelfTest to complete before  
249           reissuing the command C1.

250       c. The TPM MAY return TPM\_SUCCESS or an error code associated with C1.

251           i. This indicates that the TPM has completed the actions of TPM\_ContinueSelfTest  
252           and has completed the command C1.

253           ii. The error code MAY be TPM\_FAILEDSELFTEST.

## 254   **Actions**

255   1. If TPM\_PERMANENT\_FLAGS -> FIPS is TRUE or TPM\_PERMANENT\_FLAGS -> TPMpost  
256    is TRUE

257       a. The TPM MUST run all self-tests

258   2. Else

259       a. The TPM MUST complete all self-tests that are outstanding

260           i. Instead of completing all outstanding self-tests the TPM MAY run all self-tests

261   3. The TPM either

262       a. MAY immediately return TPM\_SUCCESS

263           i. When TPM\_ContinueSelfTest finishes execution, it MUST NOT respond to the  
264           caller with a return code.

265       b. MAY complete the self-test and then return TPM\_SUCCESS or  
266       TPM\_FAILEDSELFTEST.

### 4.3 TPM\_GetTestResult

**Start of informative comment:**

TPM\_GetTestResult provides manufacturer specific information regarding the results of the self-test. This command will work when the TPM is in self-test failure mode. The reason for allowing this command to operate in the failure mode is to allow TPM manufacturers to obtain diagnostic information.

**End of informative comment.**

#### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_GetTestResult

#### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_GetTestResult
4	4	3S	4	UINT32	outDataSize	The size of the outData area
5	<>	4S	<>	BYTE[]	outData	The outData this is manufacturer specific

#### Description

This command will work when the TPM is in self test failure mode.

#### Actions

1. The TPM SHALL respond to this command with a manufacturer specific block of information that describes the result of the latest self-test
2. The information MUST NOT contain any data that uniquely identifies an individual TPM.

## 5. Admin Opt-in

### 5.1 TPM\_SetOwnerInstall

#### Start of informative comment:

When enabled but without an owner this command sets the PERMANENT flag that allows or disallows the ability to insert an owner.

#### End of informative comment.

#### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetOwnerInstall
4	1	2S	1	BOOL	state	State to which ownership flag is to be set.

#### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetOwnerInstall

#### Action

1. If the TPM has a current owner, this command immediately returns with TPM\_SUCCESS.
2. The TPM validates the assertion of physical access. The TPM then sets the value of TPM\_PERMANENT\_FLAGS -> ownership to the value in state.

**5.2 TPM\_OwnerSetDisable****Start of informative comment:**

The TPM owner sets the PERMANENT disable flag

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OwnerSetDisable
4	1	2S	1	BOOL	disableState	Value for disable state –enable if TRUE
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
8	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner authentication. HMAC key: ownerAuth.

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OwnerSetDisable
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

**Action**

1. The TPM SHALL authenticate the command as coming from the TPM Owner. If unsuccessful, the TPM SHALL return TPM\_AUTHFAIL.
2. The TPM SHALL set the TPM\_PERMANENT\_FLAGS -> disable flag to the value in the disableState parameter.

### 5.3 TPM\_PhysicalEnable

#### Start of informative comment:

Sets the PERMANENT disable flag to FALSE using physical presence as authorization.

#### End of informative comment.

#### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PhysicalEnable

#### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PhysicalEnable

#### Action

1. Validate that physical presence is being asserted, if not return TPM\_BAD\_PRESENCE
2. The TPM SHALL set the TPM\_PERMANENT\_FLAGS.disable value to FALSE.



**5.4 TPM\_PhysicalDisable****Start of informative comment:**

Sets the PERMANENT disable flag to TRUE using physical presence as authorization

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PhysicalDisable

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PhysicalDisable

**Action**

1. Validate that physical presence is being asserted, if not return TPM\_BAD\_PRESENCE
2. The TPM SHALL set the TPM\_PERMANENT\_FLAGS.disable value to TRUE.

## 5.5 TPM\_PhysicalSetDeactivated

### Start of informative comment:

Enables the TPM using physical presence as authorization.

This command is not available when the TPM is disabled.

### End of informative comment.

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PhysicalSetDeactivated
4	1	2S	1	BOOL	state	State to which deactivated flag is to be set.

### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PhysicalSetDeactivated

### Action

1. Validate that physical presence is being asserted, if not return TPM\_BAD\_PRESENCE
2. The TPM SHALL set the TPM\_PERMANENT\_FLAGS.deactivated flag to the value in the state parameter.

## 5.6 TPM\_SetTempDeactivated

### Start of informative comment:

This command allows the operator of the platform to deactivate the TPM until the next boot of the platform.

This command requires operator authentication. The operator can provide the authentication by either the assertion of physical presence or presenting the operator AuthData value.

### End of informative comment.

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetTempDeactivated
4	4		4	TPM_AUTHHANDLE	authHandle	Auth handle for operation validation. Session type MUST be OIAP
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
5	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
7	20			TPM_AUTHDATA	operatorAuth	HMAC key: operatorAuth

### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetTempDeactivated
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: operatorAuth.

### Action

1. If tag = TPM\_TAG\_REQ\_AUTH1\_COMMAND
  - a. If TPM\_PERMANENT\_FLAGS -> operator is FALSE return TPM\_NOOPERATOR
  - b. Validate command and parameters using operatorAuth, on error return TPM\_AUTHFAIL
2. Else

- 351       a. If physical presence is not asserted the TPM MUST return TPM\_BAD\_PRESENCE
- 352    3. The TPM SHALL set the TPM\_STCLEAR\_FLAGS.deactivated flag to the value TRUE.

**5.7 TPM\_SetOperatorAuth****Start of informative comment:**

This command allows the setting of the operator AuthData value.

There is no confidentiality applied to the operator authentication as the value is sent under the assumption of being local to the platform. If there is a concern regarding the path between the TPM and the keyboard then unless the keyboard is using encryption and a secure channel an attacker can read the values.

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetOperatorAuth
4	20	2S	20	TPM_SECRET	operatorAuth	The operator AuthData

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetOperatorAuth

**Action**

1. If physical presence is not asserted the TPM MUST return TPM\_BAD\_PRESENCE
2. The TPM SHALL set the TPM\_PERMANENT\_DATA -> operatorAuth
3. The TPM SHALL set TPM\_PERMANENT\_FLAGS -> operator to TRUE

## 6. Admin Ownership

### 6.1 TPM\_TakeOwnership

**Start of informative comment:**

This command inserts the TPM Ownership value into the TPM.

**End of informative comment.**

#### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_TakeOwnership
4	2	2S	2	TPM_PROTOCOL_ID	protocolID	The ownership protocol in use.
5	4	3S	4	UINT32	encOwnerAuthSize	The size of the encOwnerAuth field
6	◇	4S	◇	BYTE[]	encOwnerAuth	The owner AuthData encrypted with PUBEK
7	4	5S	4	UINT32	encSrAuthSize	The size of the encSrAuth field
8	◇	6S	◇	BYTE[]	encSrAuth	The SRK AuthData encrypted with PUBEK
9	◇	7S	◇	TPM_KEY	srkParams	Structure containing all parameters of new SRK. pubKey.keyLength & encSize are both 0. This structure MAY be TPM_KEY12.
10	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for this command
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
11	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
12	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
13	20			TPM_AUTHDATA	ownerAuth	Authorization session digest for input params. HMAC key: the new ownerAuth value. See actions for validation operations

374

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_TakeOwnership
4	<>	3S	<>	TPM_KEY	srkPub	Structure containing all parameters of new SRK. srkPub.encData is set to 0. This structure MAY be TPM_KEY12.
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: the new ownerAuth value

375

**Description**

376

377

The type of the output srkPub MUST be the same as the type of the input srkParams, either both TPM\_KEY or both TPM\_KEY12.

378

**Actions**

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

1. If TPM\_PERMANENT\_DATA -> ownerAuth is valid return TPM\_OWNER\_SET
2. If TPM\_PERMANENT\_FLAGS -> ownership is FALSE return TPM\_INSTALL\_DISABLED
3. If TPM\_PERMANENT\_DATA -> endorsementKey is invalid return TPM\_NO\_ENDORSEMENT
4. Verify that authHandle is of type OIAP on error return TPM\_AUTHFAIL
5. Create A1 a TPM\_SECRET by decrypting encOwnerAuth using PRIVEK as the key
  - a. This requires that A1 was encrypted using the PUBEK
  - b. Validate that A1 is a length of 20 bytes, on error return TPM\_BAD\_KEY\_PROPERTY
6. Validate the command and parameters using A1 and ownerAuth, on error return TPM\_AUTHFAIL
7. Validate srkParams
  - a. If srkParams -> keyUsage is not TPM\_KEY\_STORAGE return TPM\_INVALID\_KEYUSAGE
  - b. If srkParams -> migratable is TRUE return TPM\_INVALID\_KEYUSAGE
  - c. If srkParams -> algorithmParms -> algorithmID is NOT TPM\_ALG\_RSA return TPM\_BAD\_KEY\_PROPERTY
  - d. If srkParams -> algorithmParms -> encScheme is NOT TPM\_ES\_RSAESOAEP\_SHA1\_MGF1 return TPM\_BAD\_KEY\_PROPERTY
  - e. If srkParams -> algorithmParms -> sigScheme is NOT TPM\_SS\_NONE return TPM\_BAD\_KEY\_PROPERTY

399 f. If srkParams -> algorithmParms -> parms -> keyLength MUST be greater than or  
400 equal to 2048, on error return TPM\_BAD\_KEY\_PROPERTY  
401 g. If TPM\_PERMANENT\_FLAGS -> FIPS is TRUE  
402 i. If srkParams -> authDataUsage specifies TPM\_AUTH\_NEVER return  
403 TPM\_NOTFIPS  
404 8. Generate K1 according to the srkParams on error return TPM\_BAD\_KEY\_PROPERTY  
405 9. Create A2 a TPM\_SECRET by decrypting encSrkJAuth using the PRIVEK  
406 a. This requires A2 to be encrypted using the PUBEK  
407 b. Validate that A2 is a length of 20 bytes, on error return TPM\_BAD\_KEY\_PROPERTY  
408 c. Store A2 in K1 -> usageAuth  
409 10. Store K1 in TPM\_PERMANENT\_DATA -> srk  
410 11. Store A1 in TPM\_PERMANENT\_DATA -> ownerAuth  
411 12. Create TPM\_PERMANENT\_DATA -> contextKey according to the rules for the algorithm  
412 in use by the TPM to save context blobs  
413 13. Create TPM\_PERMANENT\_DATA -> delegateKey according to the rules for the algorithm  
414 in use by the TPM to save delegate blobs  
415 14. Create TPM\_PERMANENT\_DATA -> tpmProof by using the TPM RNG  
416 15. Export TPM\_PERMANENT\_DATA -> srk as srkJPub  
417 16. Set TPM\_PERMANENT\_FLAGS -> readPubek to FALSE  
418 17. Calculate resAuth using the newly established TPM\_PERMANENT\_DATA -> ownerAuth



## 6.2 TPM\_OwnerClear

**Start of informative comment:**

The TPM\_OwnerClear command performs the clear operation under Owner authentication. This command is available until the Owner executes the TPM\_DisableOwnerClear, at which time any further invocation of this command returns TPM\_CLEAR\_DISABLED.

All state in the TPM should be cleared when the command TPM\_OwnerClear is invoked.

**End of informative comment.**

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OwnerClear
4	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
5	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Ignored
7	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner authentication. HMAC key: ownerAuth.

### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OwnerClear
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Fixed value FALSE
6	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: old ownerAuth.

### Actions

1. Verify that the TPM Owner authorizes the command and all of the input, on error return TPM\_AUTHFAIL.
2. If TPM\_PERMANENT\_FLAGS -> disableOwnerClear is TRUE then return TPM\_CLEAR\_DISABLED.
3. Unload all loaded keys.

- 434 4. The TPM MUST NOT modify the following TPM\_PERMANENT\_DATA items
- 435 a. endorsementKey
- 436 b. revMajor
- 437 c. revMinor
- 438 d. manuMaintPub
- 439 e. auditMonotonicCounter
- 440 f. monotonicCounter
- 441 g. pcrAttrib
- 442 h. rngState
- 443 i. EKReset
- 444 j. maxNVBufSize
- 445 k. lastFamilyID
- 446 l. tpmDAASeed
- 447 m. authDIR[0]
- 448 5. The TPM MUST invalidate the following TPM\_PERMANENT\_DATA items and any internal
- 449 resources associated with these items
- 450 a. ownerAuth
- 451 b. srk
- 452 c. delegateKey
- 453 d. delegateTable
- 454 e. contextKey
- 455 f. tpmProof
- 456 g. operatorAuth
- 457 6. The TPM MUST reset to manufacturing defaults the following TPM\_PERMANENT\_DATA
- 458 items
- 459 a. noOwnerNVWrite MUST be set to 0
- 460 b. ordinalAuditStatus
- 461 c. restrictDelegate
- 462 7. The TPM MUST invalidate or reset all fields of TPM\_STANY\_DATA
- 463 a. Nonces SHALL be reset
- 464 b. Lists (e.g. contextList) SHALL be invalidated
- 465 8. The TPM MUST invalidate or reset all fields of TPM\_STCLEAR\_DATA
- 466 a. Nonces SHALL be reset
- 467 b. Lists (e.g. contextList) SHALL be invalidated
- 468 9. The TPM MUST set the following TPM\_PERMANENT\_FLAGS to their default values

- 469       a. disable
- 470       b. deactivated
- 471       c. readPubek
- 472       d. disableOwnerClear
- 473   10. The TPM MUST set the following TPM\_PERMANENT\_FLAGS
- 474       a. ownership to TRUE
- 475       b. operator to FALSE
- 476       c. maintenanceDone to FALSE
- 477       d. allowMaintenance to TRUE
- 478   11. The TPM releases all TPM\_PERMANENT\_DATA -> monotonicCounter settings
- 479       a. This includes invalidating all currently allocated counters. The result will be no
- 480       currently allocated counters and the new owner will need to allocate counters. The
- 481       actual count value will continue to increase.
- 482   12. The TPM MUST deallocate all defined NV storage areas where
- 483       TPM\_NV\_PER\_OWNERWRITE is TRUE and nvIndex does not have the "D" bit set and
- 484       MUST NOT deallocate any other currently defined NV storage areas.
- 485   13. The TPM MUST invalidate all familyTable entries
- 486   14. The TPM MUST terminate all OSAP, DSAP, and transport sessions.
- 487   15. The TPM MUST terminate all sessions, active or saved

## 6.3 TPM\_ForceClear

### Start of informative comment:

The TPM\_ForceClear command performs the Clear operation under physical access. This command is available until the execution of the TPM\_DisableForceClear, at which time any further invocation of this command returns TPM\_CLEAR\_DISABLED.

### End of informative comment.

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ForceClear

### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ForceClear

### Actions

1. The TPM SHALL check for the assertion of physical presence, if not present return TPM\_BAD\_PRESENCE
2. If TPM\_STCLEAR\_FLAGS -> disableForceClear is TRUE return TPM\_CLEAR\_DISABLED
3. The TPM SHALL execute the actions of TPM\_OwnerClear (except for the TPM Owner authentication check)

## 6.4 TPM\_DisableOwnerClear

### Start of informative comment:

The TPM\_DisableOwnerClear command disables the ability to execute the TPM\_OwnerClear command permanently. Once invoked the only method of clearing the TPM will require physical access to the TPM.

After the execution of TPM\_ForceClear, ownerClear is re-enabled and must be explicitly disabled again by the new TPM Owner.

### End of informative comment.

## Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DisableOwnerClear
4	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
5	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
7	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner authentication. HMAC key: ownerAuth.

## Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DisableOwnerClear
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

## Actions

1. The TPM verifies that the authHandle properly authorizes the owner.
2. The TPM sets the TPM\_PERMANENT\_FLAGS -> disableOwnerClear flag to TRUE.

- 515 3. When this flag is TRUE the only mechanism that can clear the TPM is the  
516 TPM\_ForceClear command. The TPM\_ForceClear command requires physical access to  
517 the TPM to execute.

**6.5 TPM\_DisableForceClear**

Start of informative comment:

The TPM\_DisableForceClear command disables the execution of the TPM\_ForceClear command until the next startup cycle. Once this command is executed, the TPM\_ForceClear is disabled until another startup cycle is run.

End of informative comment.

**Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DisableForceClear

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DisableForceClear

**Actions**

1. The TPM sets the TPM\_STCLEAR\_FLAGS.disableForceClear flag in the TPM that disables the execution of the TPM\_ForceClear command.

## 6.6 TSC\_PhysicalPresence

### Start of informative comment:

Some TPM operations require the indication of a human's physical presence at the platform. The presence of the human either provides another indication of platform ownership or a mechanism to ensure that the execution of the command is not the result of a remote software process.

This command allows a process on the platform to indicate the assertion of physical presence. As this command is executable by software there must be protections against the improper invocation of this command.

The physicalPresenceHwEnable and physicalPresenceCmdEnable indicate the ability for either SW or HW to indicate physical presence. These flags can be reset until the physicalPresenceLifetimeLock is set. The platform manufacturer should set these flags to indicate the capabilities of the platform the TPM is bound to.

The command provides two sets of functionality. The first is to enable, permanently, either the HW or the SW ability to assert physical presence. The second is to allow SW, if enabled, to assert physical presence.

### End of informative comment.

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TSC_ORD_PhysicalPresence.
4	2	2S	2	TPM_PHYSICAL_PRESENCE	physicalPresence	The state to set the TPM's Physical Presence flags.

### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TSC_ORD_PhysicalPresence.

### Actions

- For documentation ease, the bits break into two categories. The first is the lifetime settings and the second is the assertion settings.

- Define A1 to be the lifetime settings: TPM\_PHYSICAL\_PRESENCE\_LIFETIME\_LOCK, TPM\_PHYSICAL\_PRESENCE\_HW\_ENABLE, TPM\_PHYSICAL\_PRESENCE\_CMD\_ENABLE, TPM\_PHYSICAL\_PRESENCE\_HW\_DISABLE, and TPM\_PHYSICAL\_PRESENCE\_CMD\_DISABLE



- b. Define A2 to be the assertion settings: TPM\_PHYSICAL\_PRESENCE\_LOCK, TPM\_PHYSICAL\_PRESENCE\_PRESENT, and TPM\_PHYSICAL\_PRESENCE\_NOTPRESENT

## Lifetime lock settings

### 2. If any A1 setting is present

- a. If TPM\_PERMANENT\_FLAGS -> physicalPresenceLifetimeLock is TRUE, return TPM\_BAD\_PARAMETER
- b. If any A2 setting is present return TPM\_BAD\_PARAMETER
- c. If both physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_HW\_ENABLE and physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_HW\_DISABLE are TRUE, return TPM\_BAD\_PARAMETER.
- d. If both physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_CMD\_ENABLE and physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_CMD\_DISABLE are TRUE, return TPM\_BAD\_PARAMETER.
- e. If physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_HW\_ENABLE is TRUE Set TPM\_PERMANENT\_FLAGS -> physicalPresenceHWEEnable to TRUE
- f. If physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_HW\_DISABLE is TRUE Set TPM\_PERMANENT\_FLAGS -> physicalPresenceHWEEnable to FALSE
- g. If physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_CMD\_ENABLE is TRUE, Set TPM\_PERMANENT\_FLAGS -> physicalPresenceCMDEnable to TRUE.
- h. If physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_CMD\_DISABLE is TRUE, Set TPM\_PERMANENT\_FLAGS -> physicalPresenceCMDEnable to FALSE.
- i. If physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_LIFETIME\_LOCK is TRUE
  - i. Set TPM\_PERMANENT\_FLAGS -> physicalPresenceLifetimeLock to TRUE
- j. Return TPM\_SUCCESS

## SW physical presence assertion

### 3. If any A2 setting is present

- a. If any A1 setting is present return TPM\_BAD\_PARAMETER
  - i. This check here just for consistency, the prior checks would have already ensured that this was ok
- b. If TPM\_PERMANENT\_FLAGS -> physicalPresenceCMDEnable is FALSE, return TPM\_BAD\_PARAMETER
- c. If both physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_LOCK and physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_PRESENT are TRUE, return TPM\_BAD\_PARAMETER
- d. If both physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_PRESENT and physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_NOTPRESENT are TRUE, return TPM\_BAD\_PARAMETER
- e. If TPM\_STCLEAR\_FLAGS -> physicalPresenceLock is TRUE, return TPM\_BAD\_PARAMETER

593       f. If physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_LOCK is TRUE  
594           i. Set TPM\_STCLEAR\_FLAGS -> physicalPresence to FALSE  
595           ii. Set TPM\_STCLEAR\_FLAGS -> physicalPresenceLock to TRUE  
596           iii. Return TPM\_SUCCESS  
597       g. If physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_PRESENT is TRUE  
598           i. Set TPM\_STCLEAR\_FLAGS -> physicalPresence to TRUE  
599       h. If physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_NOTPRESENT is TRUE  
600           i. Set TPM\_STCLEAR\_FLAGS -> physicalPresence to FALSE  
601           i. Return TPM\_SUCCESS  
602   4. Else // There were no A1 or A2 parameters set  
603       a. Return TPM\_BAD\_PARAMETER

**6.7 TSC\_ResetEstablishmentBit****Start of informative comment:**

The PC TPM Interface Specification (TIS) specifies setting tpmEstablished to TRUE upon execution of the HASH\_START sequence. The setting implies the creation of a Trusted Operating System on the platform. Platforms will use the value of tpmEstablished to determine if operations necessary to maintain the security perimeter are necessary.

The tpmEstablished bit provides a non-volatile, secure reporting that a HASH\_START was previously run on the platform. When a platform makes use of the tpmEstablished bit, the platform can reset tpmEstablished as the operation is no longer necessary.

For example, a platform could use tpmEstablished to ensure that, if HASH\_START had ever been, executed the platform could use the value to invoke special processing. Once the processing is complete the platform will wish to reset tpmEstablished to avoid invoking the special process again.

The TPM\_PERMANENT\_FLAGS -> tpmEstablished bit described in the TPM specifications uses positive logic. The TPM\_ACCESS register uses negative logic, so that TRUE is reflected as a 0.

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TSC_ORD_ResetEstablishmentBit

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TSC_ORD_ResetEstablishmentBit

**Actions**

1. Validate the assertion of locality 3 or locality 4
2. Set TPM\_PERMANENT\_FLAGS -> tpmEstablished to FALSE
3. Return TPM\_SUCCESS

## 7. The Capability Commands

### **Start of informative comment:**

The TPM has numerous capabilities that a remote entity may wish to know about. These items include support of algorithms, key sizes, protocols and vendor-specific additions. The TPM\_GetCapability command allows the TPM to report back to the requestor what type of TPM it is dealing with.

The request for information requires the requestor to specify which piece of information that is required. The request does not allow the “merging” of multiple requests and returns only a single piece of information.

In failure mode, the TPM returns a limited set of information that includes the TPM manufacturer and version.

In version 1.2 with the deletion of TPM\_GetCapabilitySigned the way to obtain a signed listing of the capabilities is to create a transport session, perform TPM\_GetCapability commands to list the information and then close the transport session using TPM\_ReleaseTransportSigned.

### **End of informative comment.**

1. The standard information provided in TPM\_GetCapability MUST NOT provide unique information
  - a. The TPM has no control of information placed into areas on the TPM like the NV store that is reported by the TPM. Configuration information for these areas could conceivably be unique

**7.1 TPM\_GetCapability****Start of informative comment:**

This command returns current information regarding the TPM.

The limitation on what can be returned in failure mode restricts the information a manufacturer may return when capArea indicates TPM\_CAP\_MFR.

**End of informative comment.****Incoming Parameters and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetCapability
4	4	2S	4	TPM_CAPABILITY_AREA	capArea	Partition of capabilities to be interrogated
5	4	3S	4	UINT32	subCapSize	Size of subCap parameter
6	<>	4S	<>	BYTE[]	subCap	Further definition of information

**Outgoing Parameters and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetCapability
4	4	3S	4	UINT32	respSize	The length of the returned capability response
5	<>	4S	<>	BYTE[]	resp	The capability response

**Actions**

1. The TPM validates the capArea and subCap indicators. If the information is available, the TPM creates the response field and fills in the actual information.
2. The structure document contains the list of caparea and subCap values
3. If the TPM is in failure mode
  - a. The TPM MUST only return TPM manufacturer and TPM version.
4. If the TPM is in limited operation mode
  - a. The TPM MUST only return TPM\_CAP\_PROPERTY -> TPM\_CAP\_PROP\_DURATION.

## 7.2 TPM\_SetCapability

### Start of informative comment:

This command sets values in the TPM.

A setValue that is inconsistent with the capArea and subCap is considered a bad parameter.

### End of informative comment.

### Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	ordinal: TPM_ORD_SetCapability
4	4	2S	4	TPM_CAPABILITY_AREA	capArea	Partition of capabilities to be set
5	4	3S	4	UINT32	subCapSize	Size of subCap parameter
6	◇	4S	◇	BYTE[]	subCap	Further definition of information
7	4	5S	4	UINT32	setValueSize	The size of the value to set
8	◇	6S	◇	BYTE[]	setValue	The value to set
9	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
10	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
11	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
12	20			TPM_AUTHDATA	ownerAuth	Authorization. HMAC key: owner.usageAuth.

671 **Outgoing Parameters and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	ordinal: TPM_ORD_SetCapability
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	Authorization HMAC key: owner.usageAuth.

672 **Actions**

- 673 1. If tag = TPM\_TAG\_RQU\_AUTH1\_COMMAND, validate the command and parameters  
674 using ownerAuth, return TPM\_AUTHFAIL on error
- 675 2. The TPM validates the capArea and subCap indicators, including the ability to set value  
676 based on any set restrictions
- 677 3. If the capArea and subCap indicators conform with one of the entries in the structure  
678 TPM\_CAPABILITY\_AREA (Values for TPM\_SetCapability)
- 679 a. The TPM sets the relevant flag/data to the value of setValue parameter.
- 680 4. Else
- 681 a. Return the error code TPM\_BAD\_PARAMETER.

## 7.3 TPM\_GetCapabilityOwner

### Start of informative comment:

It can provide information to TPM\_GetCapabilitySigned which may result in an invalid signature.

TPM\_GetCapabilityOwner enables the TPM Owner to retrieve all the non-volatile flags and the volatile flags in a single operation. This command is deprecated, mandatory.

The flags summarize many operational aspects of the TPM. The information represented by some flags is private to the TPM Owner. So, for simplicity, proof of ownership of the TPM must be presented to retrieve the set of flags. When necessary, the flags that are not private to the Owner can be deduced by Users via other (more specific) means.

The normal TPM authentication mechanisms are sufficient to prove the integrity of the response. No additional integrity check is required.

### End of informative comment.

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetCapabilityOwner
4	4			TPM_AUTHHANDLE	authHandle	The authorization handle used for Owner authorization.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
5	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization handle
7	20			TPM_AUTHDATA	ownerAuth	The authorization digest for inputs and owner authorization. HMAC key: OwnerAuth.

### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation. See section 4.3.
		2S	4	TPM_COMMAND_CODE	ordinal	Ordinal: TPM_ORD_GetCapabilityOwner
4	4	3S	4	TPM_VERSION	version	A properly filled out version structure.
5	4	4S	4	UINT32	non_volatile_flags	The current state of the non-volatile flags.
6	4	5S	4	UINT32	volatile_flags	The current state of the volatile flags.
7	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active



9	20		TPM_AUTHDATA	resAuth	The authorization digest for the returned parameters. HMAC key: OwnerAuth.
---	----	--	--------------	---------	----------------------------------------------------------------------------

**Description**

For  $31 \geq N \geq 0$

1. Bit-N of the TPM\_PERMANENT\_FLAGS structure is the Nth bit after the opening bracket in the definition of TPM\_PERMANENT\_FLAGS in the version of the specification indicated by the parameter “version”. The bit immediately after the opening bracket is the 0<sup>th</sup> bit.
2. Bit-N of the TPM\_STCLEAR\_FLAGS structure is the Nth bit after the opening bracket in the definition of TPM\_STCLEAR\_FLAGS in the version of the specification indicated by the parameter “version”. The bit immediately after the opening bracket is the 0<sup>th</sup> bit.
3. Bit-N of non\_volatile\_flags corresponds to the Nth bit in TPM\_PERMANENT\_FLAGS, and the lsb of non\_volatile\_flags corresponds to bit0 of TPM\_PERMANENT\_FLAGS
4. Bit-N of volatile\_flags corresponds to the Nth bit in TPM\_STCLEAR\_FLAGS, and the lsb of volatile\_flags corresponds to bit0 of TPM\_STCLEAR\_FLAGS

**Actions**

1. The TPM validates that the TPM Owner authorizes the command.
2. The TPM creates the parameter non\_volatile\_flags by setting each bit to the same state as the corresponding bit in TPM\_PERMANENT\_FLAGS. Bits in non\_volatile\_flags for which there is no corresponding bit in TPM\_PERMANENT\_FLAGS are set to zero.
3. The TPM creates the parameter volatile\_flags by setting each bit to the same state as the corresponding bit in TPM\_STCLEAR\_FLAGS. Bits in volatile\_flags for which there is no corresponding bit in TPM\_STCLEAR\_FLAGS are set to zero.
4. The TPM generates the parameter “version”.
5. The TPM returns non\_volatile\_flags, volatile\_flags and version to the caller.

## 8. Auditing

### 8.1 Audit Generation

#### Start of informative comment:

The TPM generates an audit event in response to the TPM executing a function that has the audit flag set to TRUE for that function.

The TPM maintains an extended value for all audited operations.

Input audit generation occurs before the listed actions and output audit generation occurs after the listed actions.

#### End of informative comment.

#### Description

1. The TPM extends the audit digest whenever the ordinalAuditStatus is TRUE for the ordinal about to be executed. The only exception is if the ordinal about to be executed is TPM\_SetOrdinalAuditStatus. In that case, output parameter auditing is performed if the ordinalAuditStatus resulting from command execution is TRUE.
2. If the command is malformed
  - a. If the ordinal is unknown, unimplemented, or cannot be determined, no auditing is performed.
  - b. If the ordinal is known and audited, but the “above the line” parameters are malformed and the input parameter digest cannot be determined, use an input digest of all zeros.
    - i. Use an output digest of the return code and ordinal.
  - c. If the ordinal is known and audited, the “above the line” parameters are determined, but the “below the line” parameters are malformed, use an input digest of the “above the line” parameters.
    - i. Use an output digest of the return code and ordinal.
  - d. Malformed in this context means that, when breaking up a command into its parameters, there are too few or too many bytes in the command stream.
  - e. Breaking up a command in this context means only the parsing required to extract the parameters.
    - i. E.g., for parameter set comprising a UINT32 size and a BYTE[] array, the BYTE[] array should not be further parsed.

#### Actions

The TPM will execute the ordinal and perform auditing in the following manner

1. Map V1 to TPM\_STANY\_DATA
2. Map P1 to TPM\_PERMANENT\_DATA
3. If V1 -> auditDigest is NULL

- 758       a. Increment P1 -> auditMonotonicCounter by 1
- 759   4. Create A1 a TPM\_AUDIT\_EVENT\_IN structure
- 760       a. Set A1 -> inputParms to the digest of the input parameters from the command
- 761           i. Digest value according to the HMAC digest rules of the "above the line"
- 762             parameters (i.e. the first HMAC digest calculation).
- 763       b. Set A1 -> auditCount to P1 -> auditMonotonicCounter
- 764       c. Set V1 -> auditDigest to SHA-1 (V1 -> auditDigest || A1)
- 765   5. Execute command
- 766       a. Execution implies the performance of the listed actions for the ordinal.
- 767   6. Create A2 a TPM\_AUDIT\_EVENT\_OUT structure
- 768       a. Set A2 -> outputParms to the digest of the output parameters from the command
- 769           i. Digest value according to the HMAC digest rules of the "above the line"
- 770             parameters (i.e. the first HMAC digest calculation).
- 771       b. Set A2 -> auditCount to P1 -> auditMonotonicCounter
- 772       c. Set V1 -> auditDigest to SHA-1 (V1 -> auditDigest || A2)

## 8.2 Effect of audit failing after completion of a command

### **Start of informative comment:**

An operation could complete and then when the TPM attempts to audit the command the audit process could have an internal error.

With one return parameter, The TPM is unable to return both the audit failure and the command success or failure results. To indicate the audit failure, the TPM will return one of two error codes: TPM\_AUDITFAIL\_SUCCESSFUL (if the command completed successfully) or TPM\_AUDITFAIL\_UNSUCCESSFUL (if the command completed unsuccessfully).

This new functionality changes the 1.1 TPM functionality when this condition occurs.

### **End of informative comment.**

1. When after completion of an operation, and in performing the audit process, the TPM has an internal failure (unable to write, SHA-1 failure etc.) the TPM MUST set the internal TPM state such that the TPM returns the TPM\_FAILEDSELFTEST error on subsequent attempts to execute a command
2. The return code for the command uses the following rules
  - a. Command result success, Audit success -> return TPM\_SUCCESS
  - b. Command result failure, Audit success -> return command result failure
  - c. Command result success, Audit failure -> return TPM\_AUDITFAIL\_SUCCESSFUL
  - d. Command result failure, Audit failure -> return TPM\_AUDITFAIL\_UNSUCCESSFUL
3. If the TPM is permanently nonrecoverable after an audit failure, then the TPM MUST always return TPM\_FAILEDSELFTEST for every command other than TPM\_GetTestResult. This state must persist regardless of power cycling, the execution of TPM\_Init or any other actions.

**8.3 TPM\_GetAuditDigest****Start of informative comment:**

This returns the current audit digest. The external audit log has the responsibility to track the parameters that constitute the audit digest.

This value may be unique to an individual TPM. The value however will be changing at a rate set by the TPM Owner. Those attempting to use this value may find it changing without their knowledge. This value represents a very poor source of tracking uniqueness.

**End of informative comment.****Incoming Parameters and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4			TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetAuditDigest
4	4			UINT32	startOrdinal	The starting ordinal for the list of audited ordinals

**Outgoing Parameters and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	Tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4			TPM_RESULT	returnCode	The return code of the operation.
5	10			TPM_COUNTER_VALUE	counterValue	The current value of the audit monotonic counter
4	20			TPM_DIGEST	auditDigest	Log of all audited events
5	1			BOOL	more	TRUE if the output does not contain a full list of audited ordinals
5	4			UINT32	ordSize	Size of the ordinal list in bytes
6	<>			UINT32[]	ordList	List of ordinals that are audited.

**Description**

1. This command is never audited.

**Actions**

1. The TPM sets auditDigest to TPM\_STANY\_DATA -> auditDigest
2. The TPM sets counterValue to TPM\_PERMANENT\_DATA -> auditMonotonicCounter
3. The TPM creates an ordered list of audited ordinals. The list starts at startOrdinal listing each ordinal that is audited.
  - a. If startOrdinal is 0 then the first ordinal that could be audited would be TPM\_OIAP (ordinal 0x0000000A)

- 815       b. The next ordinal would be TPM\_OSAP (ordinal 0x0000000B)
- 816    4. If the ordered list does not fit in the output buffer the TPM sets more to TRUE
- 817    5. Return TPM\_STANY\_DATA -> auditDigest as auditDigest

**8.4 TPM\_GetAuditDigestSigned****Start of informative comment:**

The signing of the audit log returns the entire digest value and the list of currently audited commands.

The inclusion of the list of audited commands as an atomic operation is to tie the current digest value with the list of commands that are being audited.

**Note to future architects**

When auditing functionality is active in a TPM, it may seem logical to remove this ordinal from the active set of ordinals as the signing functionality of this command could be handled in a signed transport session. While true, this command has a secondary affect also, resetting the audit log digest. As the reset requires TPM Owner authentication, there must be some way in this command to reflect the TPM Owner wishes. By requiring that a TPM Identity key be the only key that can sign and reset, the TPM Owner's authentication is implicit in the execution of the command (TPM Identity Keys are created and controlled by the TPM Owner only). Hence, while one might want to remove an ordinal this is not one that can be removed if auditing is functional.

**End of informative comment.****Incoming Parameters and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetAuditDigestSigned
4	4			TPM_KEY_HANDLE	keyHandle	The handle of a loaded key that can perform digital signatures.
5	1	2S	1	BOOL	closeAudit	Indication if audit session should be closed
6	20	3S	20	TPM_NONCE	antiReplay	A nonce to prevent replay attacks
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for key authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
10	20			TPM_AUTHDATA	keyAuth	Authorization. HMAC key: key.usageAuth.

## 836 Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetAuditDigestSigned
4	10	3S	10	TPM_COUNTER_VALUE	counterValue	The value of the audit monotonic counter
5	20	4S	20	TPM_DIGEST	auditDigest	Log of all audited events
6	20	5S	20	TPM_DIGEST	ordinalDigest	Digest of all audited ordinals
7	4	6S	4	UINT32	sigSize	The size of the sig parameter
8	◇	7S	◇	BYTE[]	sig	The signature of the area
9	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
10	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
11	20	.		TPM_AUTHDATA	resAuth	Authorization HMAC key: key.usageAuth.

## 837 Actions

- 838 1. Validate the AuthData and parameters using keyAuth, return TPM\_AUTHFAIL on error
- 839 2. The TPM validates that the key pointed to by keyHandle has a signature scheme of  
840 TPM\_SS\_RSASSAPKCS1V15\_SHA1, return TPM\_INVALID\_KEYUSAGE on error
- 841 3. Create D1 a TPM\_SIGN\_INFO structure and set the structure defaults
  - 842 a. Set D1 -> fixed to "ADIG"
  - 843 b. Set D1 -> replay to antiReplay
  - 844 c. Create D3 a list of all audited ordinals as defined in the TPM\_GetAuditDigest  
845 UINT32[] ordList outgoing parameter
  - 846 d. Create D4 the SHA-1 of D3
  - 847 e. Set auditDigest to TPM\_STANY\_DATA -> auditDigest
  - 848 f. Set counterValue to TPM\_PERMANENT\_DATA -> auditMonotonicCounter
  - 849 g. Create D2 the concatenation of auditDigest || counterValue || D4
  - 850 h. Set D1 -> data to D2
  - 851 i. Create a digital signature of the SHA-1 of D1 by using the signature scheme for  
852 keyHandle
  - 853 j. Set ordinalDigest to D4
- 854 4. If closeAudit == TRUE
  - 855 a. If keyHandle->keyUsage is TPM\_KEY\_IDENTITY
  - 856 i. TPM\_STANY\_DATA -> auditDigest MUST be set to NULLS.



- 857       b. Else
- 858       i. Return TPM\_INVALID\_KEYUSAGE

## 8.5 TPM\_SetOrdinalAuditStatus

### Start of informative comment:

Set the audit flag for a given ordinal. Requires the authentication of the TPM Owner.

### End of informative comment.

### Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetOrdinalAuditStatus
4	4	2S	4	TPM_COMMAND_CODE	ordinalToAudit	The ordinal whose audit flag is to be set
5	1	3S	1	BOOL	auditState	Value for audit flag
6	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
7	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
9	20			TPM_AUTHDATA	ownerAuth	HMAC key: ownerAuth.

### Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetOrdinalAuditStatus
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

### Actions

1. Validate the AuthData to execute the command and the parameters
2. Validate that the ordinal points to a valid TPM ordinal, return TPM\_BADINDEX on error
  - a. Valid TPM ordinal means an ordinal that the TPM implementation supports
3. Set the non-volatile flag associated with ordinalToAudit to the value in auditState

870 **9. Administrative Functions - Management**

871 **9.1 TPM\_FieldUpgrade**

872 **Start of informative comment:**

873 The TPM needs a mechanism to allow for updating the protected capabilities once a TPM is  
874 in the field. Given the varied nature of TPM implementations there will be numerous  
875 methods of performing an upgrade of the protected capabilities. This command, when  
876 implemented, provides a manufacturer specific method of performing the upgrade.

877 The manufacturer can determine, within the listed requirements, how to implement this  
878 command. The command may be more than one command and actually a series of  
879 commands.

880 The IDL definition is to create an ordinal for the command, however the remaining  
881 parameters are manufacturer specific.

882 The policy to determine when it is necessary to perform the actions of TPM\_RevokeTrust is  
883 outside the TPM spec and determined by other TCG workgroups.

884 **End of informative comment.**

885 **IDL Definition**

886 TPM\_RESULT TPM\_FieldUpgrade(  
887 [in, out] TPM\_AUTH\* ownerAuth,  
888 ...);

889 **Type**

890 This is an optional command and a TPM is not required to implement this command in any  
891 form.

892 **Parameters**

Type	Name	Description
TPM_AUTH	ownerAuth	Authentication from TPM owner to execute command
...		Remaining parameters are manufacturer specific

893 **Descriptions**

894 The upgrade mechanisms in the TPM MUST not require the TPM to hold a global secret. The  
895 definition of global secret is a secret value shared by more than one TPM.

896 The TPME is not allowed to pre-store or use unique identifiers in the TPM for the purpose of  
897 field upgrade. The TPM MUST NOT use the endorsement key for identification or encryption  
898 in the upgrade process. The upgrade process MAY use a TPM Identity to deliver upgrade  
899 information to specific TPM's.

900 The upgrade process can only change protected capabilities.

901 The upgrade process can only access data in shielded locations where this data is necessary  
902 to validate the TPM Owner, validate the TPME and manipulate the blob

903 The TPM MUST be conformant to the TPM specification, protection profiles and security  
904 targets after the upgrade. The upgrade MAY NOT decrease the security values from the  
905 original security target.

906 The security target used to evaluate this TPM MUST include this command in the TOE.

907 When a field upgrade occurs, it is always sufficient to put the TPM into the same state as a  
908 successfully executed TPM\_RevokeTrust.

## 909 **Actions**

910 The TPM SHALL perform the following when executing the command:

- 911 1. Validate the TPM Owners AuthData to execute the command
- 912 2. Validate that the upgrade information was sent by the TPME. The validation mechanism  
913 MUST use a strength of function that is at least the same strength of function as a  
914 digital signature performed using a 2048 bit RSA key.
- 915 3. Validate that the upgrade target is the appropriate TPM model and version.
- 916 4. Process the upgrade information and update the protected capabilities
- 917 5. Set the TPM\_PERMANENT\_DATA.revMajor and TPM\_PERMANENT\_DATA.revMinor to the  
918 values indicated in the upgrade. The selection of the value is a manufacturer option. The  
919 values MUST be monotonically increasing. Installing an upgrade with a major and minor  
920 revision that is less than currently installed in the TPM is a valid operation.
- 921 6. Set the TPM\_STCLEAR\_FLAGS.deactivated to TRUE

**9.2 TPM\_SetRedirection****Informative comment**

The redirection command attaches a key to a redirection receiver.

When making the connection to a GPIO channel the authorization restrictions are set at connection time and not for each invocation that uses the channel.

**End of informative comments****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetRedirection
4	4			TPM_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key that can implement redirection.
5	4	2S	4	TPM_REDIR_COMMAND	redirCmd	The command to execute
6	4	3S	4	UINT32	inputDataSize	The size of the input data
7	◁	4S	◁	BYTE	inputData	Manufacturer parameter
8	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
9	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
10	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
11	20			TPM_AUTHDATA	ownerAuth	HMAC key ownerAuth

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetRedirection
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: key.usageAuth

**Action**

1. If tag == TPM\_TAG\_REQ\_AUTH1\_COMMAND

933       a. Validate the command and parameters using TPM Owner authentication, on error  
934       return TPM\_AUTHFAIL

935   2. if redirCmd == TPM\_REDIR\_GPIO

936       a. Validate that keyHandle points to a loaded key, return TPM\_INVALID\_KEYHANDLE  
937       on error

938       b. Validate the key attributes specify redirection, return TPM\_BAD\_TYPE on error

939       c. Validate that inputDataSize is 4, return TPM\_BAD\_PARAM\_SIZE on error

940       d. Validate that inputData points to a valid GPIO channel, return  
941       TPM\_BAD\_PARAMETER on error

942       e. Map C1 to the TPM\_GPIO\_CONFIG\_CHANNEL structure indicated by inputData

943       f. If C1 -> attr specifies TPM\_GPIO\_ATTR\_OWNER

944           i. If tag != TPM\_TAG\_REQ\_AUTH1\_COMMAND return TPM\_AUTHFAIL

945       g. If C1 -> attr specifies TPM\_GPIO\_ATTR\_PP

946           i. If TPM\_STCLEAR\_FLAGS -> physicalPresence == FALSE, then return  
947           TPM\_BAD\_PRESENCE

948       h. Return TPM\_SUCCESS

949   3. The TPM MAY support other redirection types. These types may be specified by TCG or  
950   provided by the manufacturer.

**9.3 TPM\_ResetLockValue****Informative comment**

Command that resets the TPM dictionary attack mitigation values

This allows the TPM owner to cancel the effect of a number of successive authorization failures. Dictionary attack mitigation is vendor specific, and the actions here are one possible implementation. The TPM may treat an authorization failure outside the mitigation time as a normal failure and not disable the command.

If this command itself has an authorization failure, it is blocked for the remainder of the lock out period. This prevents a dictionary attack on the owner authorization using this command.

It is understood that this command allows the TPM owner to perform a dictionary attack on other authorization values by alternating a trial and this command.

**End of informative comments****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ResetLockValue
4	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for TPM Owner authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
5	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
7	20			TPM_AUTHDATA	ownerAuth	HMAC key TPM Owner auth

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ResetLockValue
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	HMAC key: TPM Owner auth

**Action**

1. If TPM\_STCLEAR\_DATA -> disableResetLock is TRUE return TPM\_AUTHFAIL
  - a. The internal dictionary attack mechanism will set TPM\_STCLEAR\_DATA -> disableResetLock to FALSE when the timeout period expires
2. If the command and parameters validation using ownerAuth fails
  - a. Set TPM\_STCLEAR\_DATA -> disableResetLock to TRUE
  - b. Restart the TPM dictionary attack lock out period
  - c. Return TPM\_AUTHFAIL
3. Reset the internal TPM dictionary attack mitigation mechanism
  - a. The mechanism is vendor specific and can include time outs, reboots, and other mitigation strategies



## 10. Storage functions

### 10.1 TPM\_Seal

**Start of informative comment:**

The SEAL operation allows software to explicitly state the future “trusted” configuration that the platform must be in for the secret to be revealed. The SEAL operation also implicitly includes the relevant platform configuration (PCR-values) when the SEAL operation was performed. The SEAL operation uses the tpmProof value to BIND the blob to an individual TPM.

If the UNSEAL operation succeeds, proof of the platform configuration that was in effect when the SEAL operation was performed is returned to the caller, as well as the secret data. This proof may, or may not, be of interest. If the SEALED secret is used to authenticate the platform to a third party, a caller is normally unconcerned about the state of the platform when the secret was SEALED, and the proof may be of no interest. On the other hand, if the SEALED secret is used to authenticate a third party to the platform, a caller is normally concerned about the state of the platform when the secret was SEALED. Then the proof is of interest.

For example, if SEAL is used to store a secret key for a future configuration (probably to prove that the platform is a particular platform that is in a particular configuration), the only requirement is that that key can be used only when the platform is in that future configuration. Then there is no interest in the platform configuration when the secret key was SEALED. An example of this case is when SEAL is used to store a network authentication key.

On the other hand, suppose an OS contains an encrypted database of users allowed to log on to the platform. The OS uses a SEALED blob to store the encryption key for the user-database. However, the nature of SEAL is that any SW stack can SEAL a blob for any other software stack. Hence the OS can be attacked by a second OS replacing both the SEALED-blob encryption key, and the user database itself, allowing untrusted parties access to the services of the OS. To thwart such attacks, SEALED blobs include the past SW configuration. Hence, if the OS is concerned about such attacks, it may check to see whether the past configuration is one that is known to be trusted.

TPM\_Seal requires the encryption of one parameter (“Secret”). For the sake of uniformity with other commands that require the encryption of more than one parameter, the string used for XOR encryption is generated by concatenating a nonce (created during the OSAP session) with the session shared secret and then hashing the result.

**End of informative comment.**

## 012 Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Seal.
4	4			TPM_KEY_HANDLE	keyHandle	Handle of a loaded key that can perform seal operations.
5	20	2S	20	TPM_ENCAUTH	encAuth	The encrypted AuthData for the sealed data. The encryption key is the shared secret from the OSAP protocol.
6	4	3S	4	UINT32	pcrInfoSize	The size of the pcrInfo parameter. If 0 there are no PCR registers in use
7	◇	4S	◇	TPM_PCR_INFO	pcrInfo	The PCR selection information. The caller MAY use TPM_PCR_INFO_LONG.
8	4	5S	4	UINT32	inDataSize	The size of the inData parameter
9	◇	6S	◇	BYTE[]	inData	The data to be sealed to the platform and any specified PCRs
10	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization. Must be an OSAP session for this command.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
11	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
12	1	4H1	1	BOOL	continueAuthSession	Ignored
13	20			TPM_AUTHDATA	pubAuth	The authorization session digest for inputs and keyHandle. HMAC key: key.usageAuth.

## 013 Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Seal.
4	◇	3S	◇	TPM_STORED_DATA	sealedData	Encrypted, integrity -protected data object that is the result of the TPM_Seal operation.
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, fixed value of FALSE
7	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: key.usageAuth.

## 014 Descriptions

015 TPM\_Seal is used to encrypt private objects that can only be decrypted using TPM\_Unseal.

## 016 Actions

017 1. Validate the authorization to use the key pointed to by keyHandle

018 2. If the `inDataSize` is 0 the TPM returns `TPM_BAD_PARAMETER`

019 3. If the `keyUsage` field of the key indicated by `keyHandle` does not have the value  
020 `TPM_KEY_STORAGE`, the TPM must return the error code `TPM_INVALID_KEYUSAGE`.

021 4. If the `keyHandle` points to a migratable key then the TPM MUST return the error code  
022 `TPM_INVALID_KEY_USAGE`.

023 5. Determine the version of `pcrInfo`

024 a. If `pcrInfoSize` is 0

025 i. set `V1` to 1

026 b. Else

027 i. Point `X1` as `TPM_PCR_INFO_LONG` structure to `pcrInfo`

028 ii. If `X1 -> tag` is `TPM_TAG_PCR_INFO_LONG`

029 (1) Set `V1` to 2

030 iii. Else

031 (1) Set `V1` to 1

032 6. If `V1` is 1 then

033 a. Create `S1` a `TPM_STORED_DATA` structure

034 7. else

035 a. Create `S1` a `TPM_STORED_DATA12` structure

036 b. Set `S1 -> et` to `NULL`

037 8. Set `s1 -> encDataSize` to 0

038 9. Set `s1 -> encData` to `NULL`

039 10. Set `s1 -> sealInfoSize` to `pcrInfoSize`

040 11. If `pcrInfoSize` is not 0 then

041 a. if `V1` is 1 then

042 i. Validate `pcrInfo` as a valid `TPM_PCR_INFO` structure, return `TPM_BADINDEX` on  
043 error

044 ii. Set `s1 -> sealInfo -> pcrSelection` to `pcrInfo -> pcrSelection`

045 iii. Create `h1` the composite hash of the PCR selected by `pcrInfo -> pcrSelection`

046 iv. Set `s1 -> sealInfo -> digestAtCreation` to `h1`

047 v. Set `s1 -> sealInfo -> digestAtRelease` to `pcrInfo -> digestAtRelease`

048 b. else

049 i. Validate `pcrInfo` as a valid `TPM_PCR_INFO_LONG` structure, return  
050 `TPM_BADINDEX` on error

051 ii. Set `s1 -> sealInfo -> creationPCRSelection` to `pcrInfo -> creationPCRSelection`

052 iii. Set `s1 -> sealInfo -> releasePCRSelection` to `pcrInfo -> releasePCRSelection`

053       iv. Set s1 -> sealInfo -> digestAtRelease to pcrInfo -> digestAtRelease  
054       v. Set s1 -> sealInfo -> localityAtRelease to pcrInfo -> localityAtRelease  
055       vi. Create h2 the composite hash of the TPM\_STCLEAR\_DATA -> PCR selected by  
056       pcrInfo -> creationPCRSelection  
057       vii. Set s1 -> sealInfo -> digestAtCreation to h2  
058       viii. Set s1 -> sealInfo -> localityAtCreation to TPM\_STANY\_FLAGS ->  
059       localityModifier  
060   12.If authHandle indicates XOR encryption for the AuthData secrets  
061       a. Create X1 the SHA-1 of the concatenation of (authHandle -> sharedSecret ||  
062       authLastNonceEven)  
063       b. Create a1 by XOR X1 and encAuth  
064   13.Else  
065       a. Create a1 by decrypting encAuth using the algorithm indicated in the OSAP session  
066       b. Key is from authHandle -> sharedSecret  
067       c. IV is SHA-1 of (authLastNonceEven || nonceOdd)  
068   14.The TPM provides NO validation of a1. Well-known values (like NULLS) are valid and  
069       possible.  
070   15.Create s2 a TPM\_SEALED\_DATA structure  
071       a. Set s2 -> payload to TPM\_PT\_SEAL  
072       b. Set s2 -> tpmProof to TPM\_PERMANENT\_DATA -> tpmProof  
073       c. Create h3 the SHA-1 of s1  
074       d. Set s2 -> storedDigest to h3  
075       e. Set s2 -> authData to a1  
076       f. Set s2 -> dataSize to inDataSize  
077       g. Set s2 -> data to inData  
078   16.Validate that the size of s2 can be encrypted by the key pointed to by keyHandle, return  
079       TPM\_BAD\_DATASIZE on error  
080   17.Create s3 the encryption of s2 using the key pointed to by keyHandle  
081   18.Set continueAuthSession to FALSE  
082   19.Set s1 -> encDataSize to the size of s3  
083   20.Set s1 -> encData to s3  
084   21.Return s1 as sealedData

**10.2 TPM\_Unseal****Start of informative comment:**

The TPM\_Unseal operation will reveal TPM\_Seal'd data only if it was encrypted on this platform and the current configuration (as defined by the named PCR contents) is the one named as qualified to decrypt it. Internally, TPM\_Unseal accepts a data blob generated by a TPM\_Seal operation. TPM\_Unseal decrypts the structure internally, checks the integrity of the resulting data, and checks that the PCR named has the value named during TPM\_Seal. Additionally, the caller must supply appropriate AuthData for blob and for the key that was used to seal that data.

If the integrity, platform configuration and authorization checks succeed, the sealed data is returned to the caller; otherwise, an error is generated.

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal at TPM_ORD_Unseal.
4	4			TPM_KEY_HANDLE	parentHandle	Handle of a loaded key that can unseal the data.
5	◇	2S	◇	TPM_STORED_DATA	inData	The encrypted data generated by TPM_Seal.
6	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for parentHandle.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
7	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
9	20			TPM_AUTHDATA	parentAuth	The authorization session digest for inputs and parentHandle. HMAC key: parentKey.usageAuth.
10	4			TPM_AUTHHANDLE	dataAuthHandle	The authorization session handle used to authorize inData.
		2H2	20	TPM_NONCE	dataLastNonceEven	Even nonce previously generated by TPM
11	20	3H2	20	TPM_NONCE	datanonceOdd	Nonce generated by system associated with entityAuthHandle
12	1	4H2	1	BOOL	continueDataSession	Continue usage flag for dataAuthHandle.
13	20			TPM_AUTHDATA	dataAuth	The authorization session digest for the encrypted entity. HMAC key: entity.usageAuth.

## 098 Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal at TPM_ORD_Unseal.
4	4	3S	4	UINT32	secretSize	The used size of the output area for secret
5	⇔	4S	⇔	BYTE[]	secret	Decrypted data that had been sealed
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: parentKey.usageAuth.
9	20	2H2	20	TPM_NONCE	dataNonceEven	Even nonce newly generated by TPM.
		3H2	20	TPM_NONCE	dataNonceOdd	Nonce generated by system associated with dataAuthHandle
10	1	4H2	1	BOOL	continueDataSession	Continue use flag, TRUE if handle is still active
11	20			TPM_AUTHDATA	dataAuth	The authorization session digest used for the dataAuth session. HMAC key: entity.usageAuth.

## 099 Actions

1. The TPM MUST validate that parentAuth authorizes the use of the key in parentHandle, on error return TPM\_AUTHFAIL
2. If the keyUsage field of the key indicated by parentHandle does not have the value TPM\_KEY\_STORAGE, the TPM MUST return the error code TPM\_INVALID\_KEYUSAGE.
3. The TPM MUST check that the TPM\_KEY\_FLAGS -> Migratable flag has the value FALSE in the key indicated by parentHandle. If not, the TPM MUST return the error code TPM\_INVALID\_KEYUSAGE
4. Determine the version of inData .
  - a. If inData -> tag = TPM\_TAG\_STORED\_DATA12
    - i. Set V1 to 2
    - ii. Map S2 a TPM\_STORED\_DATA12 structure to inData
  - b. Else If inData -> ver = 1.1
    - i. Set V1 to 1
    - ii. Map S2 a TPM\_STORED\_DATA structure to inData
  - c. Else
    - i. Return TPM\_BAD\_VERSION
5. Create d1 by decrypting S2 -> encData using the key pointed to by parentHandle

- 117 6. Validate d1
- 118 a. d1 MUST be a TPM\_SEALED\_DATA structure
- 119 b. d1 -> tpmProof MUST match TPM\_PERMANENT\_DATA -> tpmProof
- 120 c. Set S2 -> encDataSize to 0
- 121 d. Set S2 -> encData to NULL
- 122 e. Create h1 the SHA-1 of inData
- 123 f. d1 -> storedDigest MUST match h1
- 124 g. d1 -> payload MUST be TPM\_PT\_SEAL
- 125 h. Any failure MUST return TPM\_NOTSEALED\_BLOB
- 126 7. The TPM MUST validate authorization to use d1 by checking that the HMAC calculation
- 127 using d1 -> authData as the shared secret matches the dataAuth. Return
- 128 TPM\_AUTHFAIL on mismatch.
- 129 8. If S2 -> sealInfoSize is not 0 then
- 130 a. If V1 is 1 then
- 131 i. Validate that S2 -> pcrInfo is a valid TPM\_PCR\_INFO structure
- 132 ii. Create h2 the composite hash of the PCR selected by S2 -> pcrInfo -> pcrSelection
- 133 b. If V1 is 2 then
- 134 i. Validate that S2 -> pcrInfo is a valid TPM\_PCR\_INFO\_LONG structure
- 135 ii. Create h2 the composite hash of the TPM\_STCLEAR\_DATA -> PCR selected by S2
- 136 -> pcrInfo -> releasePCRSelection
- 137 iii. Check that S2 -> pcrInfo -> localityAtRelease for TPM\_STANY\_DATA ->
- 138 localityModifier is TRUE
- 139 (1) For example if TPM\_STANY\_DATA -> localityModifier was 2 then S2 -> pcrInfo
- 140 -> localityAtRelease -> TPM\_LOC\_TWO would have to be TRUE
- 141 c. Compare h2 with S2 -> pcrInfo -> digestAtRelease, on mismatch return
- 142 TPM\_WRONGPCRVAL
- 143 9. If V1 is 2 and inData -> et specifies encryption (i.e. is not NULL) then
- 144 a. If tag is not TPM\_TAG\_RQU\_AUTH2\_COMMAND, return TPM\_AUTHFAIL
- 145 b. Verify that the authHandle session type is TPM\_PID\_OSAP, return TPM\_BAD\_MODE
- 146 on error.
- 147 c. If inData -> et is TPM\_ET\_XOR
- 148 i. Use MGF1 to create string X1 of length sealedDataSize. The inputs to MGF1 are;
- 149 authLastnonceEven, nonceOdd, "XOR", and authHandle -> sharedSecret. The
- 150 four concatenated values form the Z value that is the seed for MFG1.
- 151 ii. Create o1 by XOR of d1 -> data and X1
- 152 d. Else
- 153 i. Create o1 by encrypting d1 -> data using the algorithm indicated by inData -> et

154           ii. Key is from authHandle -> sharedSecret  
155           iii. IV is SHA-1 of (authLastNonceEven || nonceOdd)  
156       e. Set continueAuthSession to FALSE  
157   10.else  
158       a. Set o1 to d1 -> data  
159   11.Set the return secret as o1  
160   12.Return TPM\_SUCCESS



### 10.3 TPM\_UnBind

**Start of informative comment:**

TPM\_UnBind takes the data blob that is the result of a Tspi\_Data\_Bind command and decrypts it for export to the User. The caller must authorize the use of the key that will decrypt the incoming blob.

TPM\_UnBind operates on a block-by-block basis, and has no notion of any relation between one block and another.

**End of informative comment.**

#### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_UnBind.
4	4			TPM_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key that can perform UnBind operations.
5	4	2S	4	UINT32	inDataSize	The size of the input blob
6	<>	3S	<>	BYTE[]	inData	Encrypted blob to be decrypted
7	4			TPM_AUTHHANDLE	authHandle	The handle used for keyHandle authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
10	20			TPM_AUTHDATA	privAuth	The authorization session digest that authorizes the inputs and use of keyHandle. HMAC key: key.usageAuth.

## 170 Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_UnBind
4	4	3S	4	UINT32	outDataSize	The length of the returned decrypted data
5	<>	4S	<>	BYTE[]	outData	The resulting decrypted data.
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: key.usageAuth.

## 171 Description

172 TPM\_UnBind SHALL operate on a single block only.

## 173 Actions

174 The TPM SHALL perform the following:

- 175 1. If the inDataSize is 0 the TPM returns TPM\_BAD\_PARAMETER
- 176 2. Validate the AuthData to use the key pointed to by keyHandle
- 177 3. If the keyUsage field of the key referenced by keyHandle does not have the value  
178 TPM\_KEY\_BIND or TPM\_KEY\_LEGACY, the TPM must return the error code  
179 TPM\_INVALID\_KEYUSAGE
- 180 4. Decrypt the inData using the key pointed to by keyHandle
- 181 5. if (keyHandle -> encScheme does not equal TPM\_ES\_RSAESOAEP\_SHA1\_MGF1) and  
182 (keyHandle -> keyUsage equals TPM\_KEY\_LEGACY),
  - 183 a. The payload does not have TPM specific markers to validate, so no consistency check  
184 can be performed.
  - 185 b. Set the output parameter outData to the value of the decrypted value of inData.  
186 (Padding associated with the encryption wrapping of inData SHALL NOT be returned.)
  - 187 c. Set the output parameter outDataSize to the size of outData, as deduced from the  
188 decryption process.
- 189 6. else
  - 190 a. Interpret the decrypted data under the assumption that it is a TPM\_BOUND\_DATA  
191 structure, and validate that the payload type is TPM\_PT\_BIND

- 192       b. Set the output parameter outData to the value of TPM\_BOUND\_DATA ->  
193       payloadData. (Other parameters of TPM\_BOUND\_DATA SHALL NOT be returned.  
194       Padding associated with the encryption wrapping of inData SHALL NOT be returned.)
- 195       c. Set the output parameter outDataSize to the size of outData, as deduced from the  
196       decryption process and the interpretation of TPM\_BOUND\_DATA.
- 197    7. Return the output parameters.

## 10.4 TPM\_CreateWrapKey

### Start of informative comment:

The TPM\_CreateWrapKey command both generates and creates a secure storage bundle for asymmetric keys.

The newly created key can be locked to a specific PCR value by specifying a set of PCR registers.

### End of informative comment.

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateWrapKey
4	4			TPM_KEY_HANDLE	parentHandle	Handle of a loaded key that can perform key wrapping.
5	20	2S	20	TPM_ENCAUTH	dataUsageAuth	Encrypted usage AuthData for the sealed data.
6	20	3S	20	TPM_ENCAUTH	dataMigrationAuth	Encrypted migration AuthData for the sealed data.
7	<>	4S	<>	TPM_KEY	keyInfo	Information about key to be created, pubkey.keyLength and keyInfo.encData elements are 0. MAY be TPM_KEY12
8	4			TPM_AUTHHANDLE	authHandle	parent key authorization. Must be an OSAP session.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
9	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
10	1	4H1	1	BOOL	continueAuthSession	Ignored
11	20			TPM_AUTHDATA	pubAuth	Authorization HMAC key: parentKey.usageAuth.

### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateWrapKey
4	<>	4S	<>	TPM_KEY	wrappedKey	The TPM_KEY structure which includes the public and encrypted private key. MAY be TPM_KEY12
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, fixed at FALSE
7	20			TPM_AUTHDATA	resAuth	Authorization HMAC key: parentKey.usageAuth.

**207 Actions**

208 The TPM SHALL do the following:

- 209 1. Validate the AuthData to use the key pointed to by parentHandle. Return  
210 TPM\_AUTHFAIL on any error.
- 211 2. Validate the session type for parentHandle is OSAP.
- 212 3. If the TPM is not designed to create a key of the type requested in keyInfo, return the  
213 error code TPM\_BAD\_KEY\_PROPERTY
- 214 4. Verify that parentHandle->keyUsage equals TPM\_KEY\_STORAGE
- 215 5. If parentHandle -> keyFlags -> migratable is TRUE and keyInfo -> keyFlags -> migratable  
216 is FALSE then return TPM\_INVALID\_KEYUSAGE
- 217 6. Validate key parameters
  - 218 a. keyInfo -> keyUsage MUST NOT be TPM\_KEY\_IDENTITY or  
219 TPM\_KEY\_AUTHCHANGE. If it is, return TPM\_INVALID\_KEYUSAGE
  - 220 b. If keyInfo -> keyFlags -> migrateAuthority is TRUE then return  
221 TPM\_INVALID\_KEYUSAGE
- 222 7. If TPM\_PERMANENT\_FLAGS -> FIPS is TRUE then
  - 223 a. If keyInfo -> keySize is less than 1024 return TPM\_NOTFIPS
  - 224 b. If keyInfo -> authDataUsage specifies TPM\_AUTH\_NEVER return TPM\_NOTFIPS
  - 225 c. If keyInfo -> keyUsage specifies TPM\_KEY\_LEGACY return TPM\_NOTFIPS
- 226 8. If keyInfo -> keyUsage equals TPM\_KEY\_STORAGE or TPM\_KEY\_MIGRATE
  - 227 i. algorithmID MUST be TPM\_ALG\_RSA
  - 228 ii. encScheme MUST be TPM\_ES\_RSAESOAEP\_SHA1\_MGF1
  - 229 iii. sigScheme MUST be TPM\_SS\_NONE
  - 230 iv. key size MUST be 2048
- 231 9. Determine the version of key
  - 232 a. If keyInfo -> ver is 1.1
    - 233 i. Set V1 to 1
    - 234 ii. Map wrappedKey to a TPM\_KEY structure
    - 235 iii. Validate all remaining TPM\_KEY structures
  - 236 b. Else if keyInfo -> tag is TPM\_TAG\_KEY12
    - 237 i. Set V1 to 2
    - 238 ii. Map wrappedKey to a TPM\_KEY12 structure
    - 239 iii. Validate all remaining TPM\_KEY12 structures
- 240 10. If authHandle indicates XOR encryption for the AuthData secrets

241       a. Create X1 the SHA-1 of the concatenation of (authHandle -> sharedSecret ||  
242       authLastNonceEven)

243       b. Create X2 the SHA-1 of the concatenation of (authHandle -> sharedSecret ||  
244       nonceOdd)

245       c. Create DU1 the XOR of dataUsageAuth and X1

246       d. Create DM1 the XOR of dataMigrationAuth and X2

247   11.Else

248       a. Decrypt dataUsageAuth and dataMigrationAuth using the algorithm indicated in the  
249       OSAP session

250           i. Create DU1 from dataUsageAuth

251           ii. Create DM1 from dataMigrationAuth

252       b. Key is from authHandle -> sharedSecret

253       c. IV is SHA-1 of (authLastNonceEven || nonceOdd)

254   12.Set continueAuthSession to FALSE

255   13.Generate asymmetric key according to algorithm information in keyInfo

256   14.Fill in the wrappedKey structure with information from the newly generated key.

257       a. Set wrappedKey -> encData -> usageAuth to DU1

258       b. If the KeyFlags -> migratable bit is set to 1, the wrappedKey -> encData ->  
259       migrationAuth SHALL contain the decrypted value from DataMigrationAuth.

260       c. If the KeyFlags -> migratable bit is set to 0, the wrappedKey -> encData ->  
261       migrationAuth SHALL be set to the value tpmProof

262   15.If keyInfo->PCRInfoSize is non-zero

263       a. If V1 is 1

264           i. Set wrappedKey -> pcrInfo to a TPM\_PCR\_INFO structure using the pcrSelection  
265           to indicate the PCR's in use

266       b. Else

267           i. Set wrappedKey -> pcrInfo to a TPM\_PCR\_INFO\_LONG structure

268       c. Set digestAtCreation to the TPM\_COMPOSITE\_HASH indicated by  
269       creationPCRSelection

270       d. If V1 is 2 set localityAtCreation to TPM\_STANY\_DATA -> locality

271   16.Encrypt the private portions of the wrappedKey structure using the key in parentHandle

272   17.Return the newly generated key in the wrappedKey parameter

## 10.5 TPM\_LoadKey2

### Start of informative comment:

Before the TPM can use a key to either wrap, unwrap, bind, unbind, seal, unseal, sign or perform any other action, it needs to be present in the TPM. The TPM\_LoadKey2 function loads the key into the TPM for further use.

The TPM assigns the key handle. The TPM always locates a loaded key by use of the handle. The assumption is that the handle may change due to key management operations. It is the responsibility of upper level software to maintain the mapping between handle and any label used by external software.

This command has the responsibility of enforcing restrictions on the use of keys. For example, when attempting to load a STORAGE key it will be checked for the restrictions on a storage key (2048 size etc.).

The load command must maintain a record of whether any previous key in the key hierarchy was bound to a PCR using parentPCRStatus.

The flag parentPCRStatus enables the possibility of checking that a platform passed through some particular state or states before finishing in the current state. A grandparent key could be linked to state-1, a parent key could be linked to state-2, and a child key could be linked to state-3, for example. The use of the child key then indicates that the platform passed through states 1 and 2 and is currently in state 3, in this example. TPM\_Startup with stType == TPM\_ST\_CLEAR indicates that the platform has been reset, so the platform has not passed through the previous states. Hence keys with parentPCRStatus==TRUE must be unloaded if TPM\_Startup is issued with stType == TPM\_ST\_CLEAR.

If a TPM\_KEY structure has been decrypted AND the integrity test using "pubDataDigest" has passed AND the key is non-migratory, the key must have been created by the TPM. So there is every reason to believe that the key poses no security threat to the TPM. While there is no known attack from a rogue migratory key, there is a desire to verify that a loaded migratory key is a real key, arising from a general sense of unease about execution of arbitrary data as a key. Ideally a consistency check would consist of an encrypt/decrypt cycle, but this may be expensive. For RSA keys, it is therefore suggested that the consistency test consists of dividing the supposed RSA product by the supposed RSA prime, and checking that there is no remainder.

### End of informative comment.

## 305 Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadKey 2
4	4			TPM_KEY_HANDLE	parentHandle	TPM handle of parent key.
5	<>	2S	<>	TPM_KEY	inKey	Incoming key structure, both encrypted private and clear public portions. MAY be TPM_KEY12
6	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for parentHandle authorization.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
7	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
9	20			TPM_AUTHDATA	parentAuth	The authorization session digest for inputs and parentHandle. HMAC key: parentKey.usageAuth.

## 306 Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadKey 2
4	4			TPM_KEY_HANDLE	inkeyHandle	Internal TPM handle where decrypted key was loaded.
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: parentKey.usageAuth.

## 307 Actions

308 The TPM SHALL perform the following steps:

- 309 1. Validate the command and the parameters using parentAuth and parentHandle ->  
310 usageAuth
- 311 2. If parentHandle -> keyUsage is NOT TPM\_KEY\_STORAGE return  
312 TPM\_INVALID\_KEYUSAGE
- 313 3. If the TPM is not designed to operate on a key of the type specified by inKey, return the  
314 error code TPM\_BAD\_KEY\_PROPERTY
- 315 4. The TPM MUST handle both TPM\_KEY and TPM\_KEY12 structures
- 316 5. Decrypt the inKey -> privkey to obtain TPM\_STORE\_ASYMKEY structure using the key  
317 in parentHandle



- 318 6. Validate the integrity of inKey and decrypted TPM\_STORE\_ASYMKEY
- 319 a. Reproduce inKey -> TPM\_STORE\_ASYMKEY -> pubDataDigest using the fields of
- 320 inKey, and check that the reproduced value is the same as pubDataDigest
- 321 7. Validate the consistency of the key and it's key usage.
- 322 a. If inKey -> keyFlags -> migratable is TRUE, the TPM SHALL verify consistency of the
- 323 public and private components of the asymmetric key pair. If inKey -> keyFlags ->
- 324 migratable is FALSE, the TPM MAY verify consistency of the public and private
- 325 components of the asymmetric key pair. The consistency of an RSA key pair MAY be
- 326 verified by dividing the supposed (P\*Q) product by a supposed prime and checking that
- 327 there is no remainder..
- 328 b. If inKey -> keyUsage is TPM\_KEY\_IDENTITY, verify that inKey->keyFlags->migratable
- 329 is FALSE. If it is not, return TPM\_INVALID\_KEYUSAGE
- 330 c. If inKey -> keyUsage is TPM\_KEY\_AUTHCHANGE, return TPM\_INVALID\_KEYUSAGE
- 331 d. If inKey -> keyFlags -> migratable equals 0 then verify that TPM\_STORE\_ASYMKEY -
- 332 > migrationAuth equals TPM\_PERMANENT\_DATA -> tpmProof
- 333 e. Validate the mix of encryption and signature schemes
- 334 f. If TPM\_PERMANENT\_FLAGS -> FIPS is TRUE then
- 335 i. If keyInfo -> keySize is less than 1024 return TPM\_NOTFIPS
- 336 ii. If keyInfo -> authDataUsage specifies TPM\_AUTH\_NEVER return TPM\_NOTFIPS
- 337 iii. If keyInfo -> keyUsage specifies TPM\_KEY\_LEGACY return TPM\_NOTFIPS
- 338 g. If inKey -> keyUsage is TPM\_KEY\_STORAGE or TPM\_KEY\_MIGRATE
- 339 i. algorithmID MUST be TPM\_ALG\_RSA
- 340 ii. Key size MUST be 2048
- 341 iii. sigScheme MUST be TPM\_SS\_NONE
- 342 h. If inKey -> keyUsage is TPM\_KEY\_IDENTITY
- 343 i. algorithmID MUST be TPM\_ALG\_RSA
- 344 ii. Key size MUST be 2048
- 345 iii. encScheme MUST be TPM\_ES\_NONE
- 346 i. If the decrypted inKey -> pcrInfo is NULL,
- 347 i. The TPM MUST set the internal indicator to indicate that the key is not using any
- 348 PCR registers.
- 349 j. Else
- 350 i. The TPM MUST store pcrInfo in a manner that allows the TPM to calculate a
- 351 composite hash whenever the key will be in use
- 352 ii. The TPM MUST handle both version 1.1 TPM\_PCR\_INFO and 1.2
- 353 TPM\_PCR\_INFO\_LONG structures according to the type of TPM\_KEY structure
- 354 (1) The TPM MUST validate the TPM\_PCR\_INFO or TPM\_PCR\_INFO\_LONG
- 355 structures

- 356 8. Perform any processing necessary to make TPM\_STORE\_ASYMKEY key available for  
357 operations
- 358 9. Load key and key information into internal memory of the TPM. If insufficient memory  
359 exists return error TPM\_NOSPACE.
- 360 10. Assign inKeyHandle according to internal TPM rules.
- 361 11. Set InKeyHandle -> parentPCRStatus to parentHandle -> parentPCRStatus.
- 362 12. If ParentHandle indicates that it is using PCR registers, then set inKeyHandle ->  
363 parentPCRStatus to TRUE.

**10.6 TPM\_GetPubKey****Start of informative comment:**

The owner of a key may wish to obtain the public key value from a loaded key. This information may have privacy concerns so the command must have authorization from the key owner.

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_GetPubKey.
4	4			TPM_KEY_HANDLE	keyHandle	TPM handle of key.
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
8	20			TPM_AUTHDATA	keyAuth	Authorization HMAC key: key.usageAuth.

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_GetPubKey.
4	<>	3S	<>	TPM_PUBKEY	pubKey	Public portion of key in keyHandle.
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	resAuth	Authorization. HMAC key: key.usageAuth.

**Actions**

The TPM SHALL perform the following steps:

1. Validate the command the parameters using keyAuth, on error
  - a. If keyHandle has TPM\_AUTH\_PRIV\_USE\_ONLY ignore the error
  - b. Otherwise return TPM\_AUTHFAIL
2. If keyHandle == TPM\_KH\_SRK then

378       a. If TPM\_PERMANENT\_FLAGS -> readSRKPub is FALSE then return  
379       TPM\_INVALID\_KEYHANDLE  
380    3. If keyHandle -> pcrInfoSize is not 0  
381       a. If keyHandle -> keyFlags has pcrIgnoredOnRead set to FALSE  
382           i. Create a digestAtRelease according to the specified PCR registers and compare to  
383           keyHandle -> digestAtRelease and if a mismatch return TPM\_WRONGPCRVAL  
384           ii. If specified validate any locality requests  
385    4. Create a TPM\_PUBKEY structure and return

## 10.7 TPM\_SealX

### Start of informative comment:

The SEALX command works exactly like the SEAL command with the additional requirement of encryption for the inData parameter. This command also places in the sealed blob the information that the unseal also requires encryption.

SEALX requires the use of 1.2 data structures. The actions are the same as SEAL without the checks for 1.1 data structure usage.

### End of informative comment.

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SealX
4	4			TPM_KEY_HANDLE	keyHandle	Handle of a loaded key that can perform seal operations.
5	20	2S	20	TPM_ENCAUTH	encAuth	The encrypted AuthData for the sealed data. The encryption key is the shared secret from the OSAP protocol.
6	4	3S	4	UINT32	pcrInfoSize	The size of the pcrInfo parameter. If 0 there are no PCR registers in use
7	◇	4S	◇	TPM_PCR_INFO	pcrInfo	MUST use TPM_PCR_INFO_LONG.
8	4	5S	4	UINT32	inDataSize	The size of the inData parameter
9	◇	6S	◇	BYTE[]	inData	The data to be sealed to the platform and any specified PCRs
10	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization. Must be an OSAP session for this command.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
11	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
12	1	4H1	1	BOOL	continueAuthSession	Ignored
13	20			TPM_AUTHDATA	pubAuth	The authorization session digest for inputs and keyHandle. HMAC key: key.usageAuth.

## 395 Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Sealx
4	<>	3S	4	TPM_STORED_DATA	sealedData	Encrypted, integrity -protected data object that is the result of the TPM_Sealx operation.
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, fixed value of FALSE
7	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: key.usageAuth.

## 396 Actions

- 397 1. Validate the authorization to use the key pointed to by keyHandle
- 398 2. If the inDataSize is 0 the TPM returns TPM\_BAD\_PARAMETER
- 399 3. If the keyUsage field of the key indicated by keyHandle does not have the value
- 400 TPM\_KEY\_STORAGE, the TPM must return the error code TPM\_INVALID\_KEYUSAGE.
- 401 4. If the keyHandle points to a migratable key then the TPM MUST return the error code
- 402 TPM\_INVALID\_KEY\_USAGE.
- 403 5. Create s1 a TPM\_STORED\_DATA12 structure
- 404 6. Set s1 -> encDataSize to 0
- 405 7. Set s1 -> encData to NULL
- 406 8. Set s1 -> sealInfoSize to pcrInfoSize
- 407 9. If pcrInfoSize is not 0 then
  - 408 a. Validate pcrInfo as a valid TPM\_PCR\_INFO\_LONG structure, return TPM\_BADINDEX
  - 409 on error
  - 410 b. Set s1 -> sealInfo -> creationPCRSelection to pcrInfo -> creationPCRSelection
  - 411 c. Set s1 -> sealInfo -> releasePCRSelection to pcrInfo -> releasePCRSelection
  - 412 d. Set s1 -> sealInfo -> digestAtRelease to pcrInfo -> digestAtRelease
  - 413 e. Set s1 -> sealInfo -> localityAtRelease to pcrInfo -> localityAtRelease
  - 414 f. Create h2 the composite hash of the TPM\_STCLEAR\_DATA -> PCR selected by
  - 415 pcrInfo -> creationPCRSelection
  - 416 g. Set s1 -> sealInfo -> digestAtCreation to h2
  - 417 h. Set s1 -> sealInfo -> localityAtCreation to TPM\_STANY\_DATA -> localityModifier
- 418 10. Create s2 a TPM\_SEALED\_DATA structure

419 11.If authHandle indicates XOR encryption for the AuthData secrets  
420 a. Create X1 the SHA-1 of the concatenation of (authHandle -> sharedSecret ||  
421 authLastNonceEven)  
422 b. Create a1 by XOR X1 and encAuth  
423 c. Set s1 -> et to TPM\_ET\_XOR || TPM\_ET\_KEY  
424 i. TPM\_ET\_KEY is added because TPM\_Unseal uses NULL as a special value  
425 indicating no encryption.

426 12.Else  
427 a. Create a1 by decrypting encAuth using the algorithm indicated in the OSAP session  
428 b. Key is from authHandle -> sharedSecret  
429 c. IV is SHA-1 of (authLastNonceEven || nonceOdd)  
430 d. Set S1 -> et to algorithm indicated in the OSAP session

431 13.The TPM provides NO validation of a1. Well-known values (like NULLS) are valid and  
432 possible.

433 14.If authHandle indicates XOR encryption  
434 a. Use MGF1 to create string X2 of length inDataSize. The inputs to MGF1 are;  
435 authLastNonceEven, nonceOdd, "XOR", and authHandle -> sharedSecret. The four  
436 concatenated values form the Z value that is the seed for MFG1.  
437 b. Create o1 by XOR of inData and X2

438 15.Else  
439 a. Create o1 by decrypting inData using the algorithm indicated by authHandle  
440 b. Key is from authHandle -> sharedSecret  
441 c. IV is SHA-1 of (authLastNonceEven || nonceOdd)

442 16.Create s2 a TPM\_SEALED\_DATA structure  
443 a. Set s2 -> payload to TPM\_PT\_SEAL  
444 b. Set s2 -> tpmProof to TPM\_PERMANENT\_DATA -> tpmProof  
445 c. Create h3 the SHA-1 of s1  
446 d. Set s2 -> storedDigest to h3  
447 e. Set s2 -> authData to a1  
448 f. Set s2 -> dataSize to inDataSize  
449 g. Set s2 -> data to o1

450 17.Validate that the size of s2 can be encrypted by the key pointed to by keyHandle, return  
451 TPM\_BAD\_DATASIZE on error

452 18.Create s3 the encryption of s2 using the key pointed to by keyHandle

453 19.Set continueAuthSession to FALSE

454 20.Set s1 -> encDataSize to the size of s3

- 455 21.Set s1 -> encData to s3
- 456 22.Return s1 as sealedData



## 11. Migration

### Start of informative comment:

The migration of a key from one TPM to another is a vital aspect to many use models of the TPM. The migration commands are the commands that allow this operation to occur.

There are two types of migratable keys, the version 1.1 migratable keys and the version 1.2 certifiable migratable keys.

### End of informative comment.

## 11.1 TPM\_CreateMigrationBlob

### Start of informative comment:

The TPM\_CreateMigrationBlob command implements the first step in the process of moving a migratable key to a new parent or platform. Execution of this command requires knowledge of the migrationAuth field of the key to be migrated.

Migrate mode is generally used to migrate keys from one TPM to another for backup, upgrade or to clone a key on another platform. To do this, the TPM needs to create a data blob that another TPM can deal with. This is done by loading in a backup public key that will be used by the TPM to create a new data blob for a migratable key.

The TPM Owner does the selection and authorization of migration public keys at any time prior to the execution of TPM\_CreateMigrationBlob by performing the TPM\_AuthorizeMigrationKey command.

IReWrap mode is used to directly move the key to a new parent (either on this platform or another). The TPM simply re-encrypts the key using a new parent, and outputs a normal encrypted element that can be subsequently used by a TPM\_LoadKey command.

TPM\_CreateMigrationBlob implicitly cannot be used to migrate a non-migratory key. No explicit check is required. Only the TPM knows tpmProof. Therefore it is impossible for the caller to submit an AuthData value equal to tpmProof and migrate a non-migratory key.

### End of informative comment.

## 483 Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateMigrationBlob
4	4			TPM_KEY_HANDLE	parentHandle	Handle of the parent key that can decrypt encData.
5	2	2S	2	TPM_MIGRATE_SCHEME	migrationType	The migration type, either MIGRATE or REWRAP
6	◇	3S	◇	TPM_MIGRATIONKEYAUTH	migrationKeyAuth	Migration public key and its authorization session digest.
7	4	4S	4	UINT32	encDataSize	The size of the encData parameter
8	◇	5S	◇	BYTE[]	encData	The encrypted entity that is to be modified.
9	4			TPM_AUTHHANDLE	parentAuthHandle	The authorization session handle used for the parent key.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
10	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with parentAuthHandle
11	1	4H1	1	BOOL	continueAuthSession	Continue use flag for parent session
12	20		20	TPM_AUTHDATA	parentAuth	Authorization HMAC key: parentKey.usageAuth.
13	4			TPM_AUTHHANDLE	entityAuthHandle	The authorization session handle used for the encrypted entity.
		2H2	20	TPM_NONCE	entitylastNonceEven	Even nonce previously generated by TPM
14	20	3H2	20	TPM_NONCE	entitynonceOdd	Nonce generated by system associated with entityAuthHandle
15	1	4H2	1	BOOL	continueEntitySession	Continue use flag for entity session
16	20			TPM_AUTHDATA	entityAuth	Authorization HMAC key: entity.migrationAuth.

484

485 **Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateMigrationBlob
4	4	3S	4	UINT32	randomSize	The used size of the output area for random
5	<>	4S	<>	BYTE[]	random	String used for xor encryption
6	4	5S	4	UINT32	outDataSize	The used size of the output area for outData
7	<>	6S	<>	BYTE[]	outData	The modified, encrypted entity.
8	20	3H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		4H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with parentAuthHandle
9	1	5H1	1	BOOL	continueAuthSession	Continue use flag for parent key session
10	20		20	TPM_AUTHDATA	resAuth	Authorization. HMAC key: parentKey.usageAuth.
11	20	3H2	20	TPM_NONCE	entityNonceEven	Even nonce newly generated by TPM to cover entity
		4H2	20	TPM_NONCE	entitynonceOdd	Nonce generated by system associated with entityAuthHandle
12	1	5 H2	1	BOOL	continueEntity Session	Continue use flag for entity session
13	20			TPM_AUTHDATA	entityAuth	Authorization HMAC key: entity.migrationAuth.

486 **Description**

487 The TPM does not check the PCR values when migrating values locked to a PCR.

488 The second authorization session (using entityAuth) MUST be OIAP because OSAP does not  
489 have a suitable entityType490 **Actions**

- 491 1. Validate that parentAuth authorizes the use of the key pointed to by parentHandle.
- 492 2. Create d1 a TPM\_STORE\_ASYMKEY structure by decrypting encData using the key  
493 pointed to by parentHandle.
- 494 a. Verify that d1 -> payload is TPM\_PT\_ASYM.
- 495 3. Validate that entityAuth authorizes the migration of d1. The validation MUST use d1 ->  
496 migrationAuth as the secret.
- 497 4. Verify that the digest within migrationKeyAuth is legal for this TPM and public key
- 498 5. If migrationType == TPM\_MS\_MIGRATE the TPM SHALL perform the following actions:
- 499 a. Build two byte arrays, K1 and K2:
- 500 i. K1 = d1.privKey[0..19] (d1.privKey.keyLength + 16 bytes of d1.privKey.key),  
501 sizeof(K1) = 20
- 502 ii. K2 = d1.privKey[20..131] (position 16-127 of d1 . privKey.key), sizeof(K2) = 112
- 503 b. Build M1 a TPM\_MIGRATE\_ASYMKEY structure

504       i. TPM\_MIGRATE\_ASYMKEY.payload = TPM\_PT\_MIGRATE  
505       ii. TPM\_MIGRATE\_ASYMKEY.usageAuth = d1.usageAuth  
506       iii. TPM\_MIGRATE\_ASYMKEY.pubDataDigest = d1.pubDataDigest  
507       iv. TPM\_MIGRATE\_ASYMKEY.partPrivKeyLen = 112 – 127.  
508       v. TPM\_MIGRATE\_ASYMKEY.partPrivKey = K2  
509   c. Create o1 (which SHALL be 198 bytes for a 2048 bit RSA key) by performing the  
510   OAEP encoding of m using OAEP parameters of  
511       i. m = M1 the TPM\_MIGRATE\_ASYMKEY structure  
512       ii. pHash = d1->migrationAuth  
513       iii. seed = s1 = K1  
514   d. Create r1 a random value from the TPM RNG. The size of r1 MUST be the size of o1.  
515   Return r1 in the Random parameter.  
516   e. Create x1 by XOR of o1 with r1  
517   f. Copy r1 into the output field “random”.  
518   g. Encrypt x1 with the migration public key included in migrationKeyAuth.  
519   6. If migrationType == TPM\_MS\_REWRAP the TPM SHALL perform the following actions:  
520       a. Rewrap the key using the public key in migrationKeyAuth, keeping the existing  
521       contents of that key.  
522       b. Set randomSize to 0 in the output parameter array  
523   7. Else  
524       a. Return TPM\_BAD\_PARAMETER

## 11.2 TPM\_ConvertMigrationBlob

**Start of informative comment:**

This command takes a migration blob and creates a normal wrapped blob. The migrated blob must be loaded into the TPM using the normal TPM\_LoadKey function.

Note that the command migrates private keys, only. The migration of the associated public keys is not specified by TPM because they are not security sensitive. Migration of the associated public keys may be specified in a platform specific specification. A TPM\_KEY structure must be recreated before the migrated key can be used by the target TPM in a TPM\_LoadKey command.

**End of informative comment.**

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ConvertMigrationBlob.
4	4			TPM_KEY_HANDLE	parentHandle	Handle of a loaded key that can decrypt keys.
5	4	2S	4	UINT32	inDataSize	Size of inData
6	◇	3S	◇	BYTE [ ]	inData	The XOR'd and encrypted key
7	4	4S	4	UINT32	randomSize	Size of random
8	◇	5S	◇	BYTE [ ]	random	Random value used to hide key data.
9	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
10	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
11	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
12	20			TPM_AUTHDATA	parentAuth	The authorization session digest that authorizes the inputs and the migration of the key in parentHandle. HMAC key: parentKey.usageAuth

## 537 Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ConvertMigrationBlob
4	4	3S	4	UINT32	outDataSize	The used size of the output area for outData
5	<>	4S	<>	BYTE[]	outData	The encrypted private key that can be loaded with TPM_LoadKey
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: parentKey.usageAuth

## 538 Action

539 The TPM SHALL perform the following:

- 540 1. Validate the AuthData to use the key in parentHandle
- 541 2. If the keyUsage field of the key referenced by parentHandle does not have the value  
542 TPM\_KEY\_STORAGE, the TPM must return the error code TPM\_INVALID\_KEYUSAGE
- 543 3. Create d1 by decrypting the inData area using the key in parentHandle
- 544 4. Create o1 by XOR d1 and random parameter
- 545 5. Create m1 a TPM\_MIGRATE\_ASYMKEY structure, seed and pHash by OAEP decoding o1
- 546 6. Create k1 by combining seed and the TPM\_MIGRATE\_ASYMKEY -> partPrivKey field
- 547 7. Create d2 a TPM\_STORE\_ASYMKEY structure
  - 548 a. Verify that m1 -> payload == TPM\_PT\_MIGRATE
  - 549 b. Set d2 -> payload = TPM\_PT\_ASYM
  - 550 c. Set d2 -> usageAuth to m1 -> usageAuth
  - 551 d. Set d2 -> migrationAuth to pHash
  - 552 e. Set d2 -> pubDataDigest to m1 -> pubDataDigest
  - 553 f. Set d2 -> privKey field to k1
- 554 8. Create outData using the key in parentHandle to perform the encryption

## 11.3 TPM\_AuthorizeMigrationKey

### Start of informative comment:

This command creates an authorization blob, to allow the TPM owner to specify which migration facility they will use and allow users to migrate information without further involvement with the TPM owner.

It is the responsibility of the TPM Owner to determine whether migrationKey is appropriate for migration. The TPM checks just the cryptographic strength of migrationKey.

### End of informative comment.

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_AuthorizeMigrationKey
4	2	2S	2	TPM_MIGRATE_SCHEME	migrationScheme	Type of migration operation that is to be permitted for this key.
4	<>	3S	<>	TPM_PUBKEY	migrationKey	The public key to be authorized.
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
8	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner authorization. HMAC key: ownerAuth.

### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_AuthorizeMigrationKey
4	<>	3S	<>	TPM_MIGRATIONKEYAUTH	outData	Returned public key and authorization session digest.
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

### Action

The TPM SHALL perform the following:

- 568 1. Check that the cryptographic strength of migrationKey is at least that of a 2048 bit RSA  
569 key. If migrationKey is an RSA key, this means that migrationKey MUST be 2048 bits or  
570 greater
- 571 2. Validate the AuthData to use the TPM by the TPM Owner
- 572 3. Create a f1 a TPM\_MIGRATIONKEYAUTH structure
- 573 4. Verify that migrationKey-> algorithmParms -> encScheme is  
574 TPM\_ES\_RSAESOAEP\_SHA1\_MGF1, and return the error code  
575 TPM\_INAPPROPRIATE\_ENC if it is not
- 576 5. Set f1 -> migrationKey to the input migrationKey
- 577 6. Set f1 -> migrationScheme to the input migrationScheme
- 578 7. Create v1 by concatenating (migrationKey || migrationScheme ||  
579 TPM\_PERMANENT\_DATA -> tpmProof)
- 580 8. Create h1 by performing a SHA1 hash of v1
- 581 9. Set f1 -> digest to h1
- 582 10. Return f1 as outData



## 11.4 TPM\_MigrateKey

### Start of informative comment:

The TPM\_MigrateKey command performs the function of a migration authority.

The command is relatively simple; it just decrypts the input packet (coming from TPM\_CreateMigrationBlob or TPM\_CMK\_CreateBlob) and then re-encrypts it with the input public key. The output of this command would then be sent to TPM\_ConvertMigrationBlob or TPM\_CMK\_ConvertMigration on the target TPM.

TPM\_MigrateKey does not make ANY assumptions about the contents of the encrypted blob. Since it does not have the XOR string, it cannot actually determine much about the key that is being migrated.

This command exists to permit the TPM to be a migration authority. If used in this way, it is expected that the physical security of the system containing the TPM and the AuthData value for the MA key would be tightly controlled.

To prevent the execution of this command using any other key as a parent key, this command works only if keyUsage for maKeyHandle is TPM\_KEY\_MIGRATE.

### End of informative comment.

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_MigrateKey
4	4			TPM_KEY_HANDLE	maKeyHandle	Handle of the key to be used to migrate the key.
5	◇	2S	◇	TPM_PUBKEY	pubKey	Public key to which the blob is to be migrated
6	4	3S	4	UINT32	inDataSize	The size of inData
7	◇	4S	◇	BYTE[]	inData	The input blob
8	4			TPM_AUTHHANDLE	maAuthHandle	The authorization session handle used for maKeyHandle.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
9	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with certAuthHandle
10	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
11	20			TPM_AUTHDATA	keyAuth	The authorization session digest for the inputs and key to be signed. HMAC key: maKeyHandle.usageAuth.

## 600 Outgoing Operands and Sizes

Param		HMAC		Type	Name	Description
#	Sz	#	Sz			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_MigrateKey
4	4	3S	4	UINT32	outDataSize	The used size of the output area for outData
5	<>	4S	<>	BYTE[]	outData	The re-encrypted blob
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with certAuthHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag for cert key session
8	20			TPM_AUTHDATA	keyAuth	The authorization session digest for the target key. HMAC key: maKeyHandle.usageAuth

## 601 Actions

- 602 1. Validate that keyAuth authorizes the use of the key pointed to by maKeyHandle
- 603 2. The TPM validates that the key pointed to by maKeyHandle has a key usage value of  
604 TPM\_KEY\_MIGRATE, and that the allowed encryption scheme is  
605 TPM\_ES\_RSAESOAEP\_SHA1\_MGF1.
- 606 3. The TPM validates that pubKey is of a size supported by the TPM and that its size is  
607 consistent with the input blob and maKeyHandle.
- 608 4. The TPM decrypts inData and re-encrypts it using pubKey.

**11.5 TPM\_CMK\_SetRestrictions****Start of informative comment:**

This command is used by the Owner to dictate the usage of a certified-migration key with delegated authorization (authorization other than actual owner authorization).

This command is provided for privacy reasons and must not itself be delegated, because a certified-migration-key may involve a contractual relationship between the Owner and an external entity.

Since restrictions are validated at DSAP session use, there is no need to invalidate DSAP sessions when the restriction value changes.

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes incl. paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Ordinal: TPM_ORD_CMK_SetRestrictions
4	4	2S	4	TPM_CMK_DELEGATE	restriction	The bit mask of how to set the restrictions on CMK keys
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle TPM Owner authentication
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
8	20			TPM_AUTHDATA	ownerAuth	The authorization session digest. HMAC key:ownerAuth

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	Ordinal: TPM_ORD_CMK_SetRestrictions
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	Authorization HMAC key: ownerAuth.

**Description**

TPM\_PERMANENT\_DATA -> restrictDelegate is used as follows

623 1. If the session type is TPM\_PID\_DSAP and TPM\_KEY -> keyFlags -> migrateAuthority is  
624 TRUE  
625 a. If  
626 TPM\_KEY\_USAGE is TPM\_KEY\_SIGNING and restrictDelegate ->  
627 TPM\_CMK\_DELEGATE\_SIGNING is TRUE, or  
628 TPM\_KEY\_USAGE is TPM\_KEY\_STORAGE and restrictDelegate ->  
629 TPM\_CMK\_DELEGATE\_STORAGE is TRUE, or  
630 TPM\_KEY\_USAGE is TPM\_KEY\_BIND and restrictDelegate -> TPM\_CMK\_DELEGATE\_BIND  
631 is TRUE, or  
632 TPM\_KEY\_USAGE is TPM\_KEY\_LEGACY and restrictDelegate ->  
633 TPM\_CMK\_DELEGATE\_LEGACY is TRUE, or  
634 TPM\_KEY\_USAGE is TPM\_KEY\_MIGRATE and restrictDelegate ->  
635 TPM\_CMK\_DELEGATE\_MIGRATE is TRUE  
636 then the key can be used.  
637 b. Else return TPM\_INVALID\_KEYUSAGE.

638 **Actions**

639 1. Validate the ordinal and parameters using TPM Owner authentication, return  
640 TPM\_AUTHFAIL on error  
641 2. Set TPM\_PERMANENT\_DATA -> TPM\_CMK\_DELEGATE -> restrictDelegate = restriction  
642 3. Return TPM\_SUCCESS

**11.6 TPM\_CMK\_ApproveMA****Start of informative comment:**

This command creates an authorization ticket, to allow the TPM owner to specify which Migration Authorities they approve and allow users to create certified-migration-keys without further involvement with the TPM owner.

It is the responsibility of the TPM Owner to determine whether a particular Migration Authority is suitable to control migration

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CMK_ApproveMA
4	20	2S	20	TPM_DIGEST	migrationAuthorityDigest	A digest of a TPM_MSA_COMPOSITE structure (itself one or more digests of public keys belonging to migration authorities)
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
8	20			TPM_AUTHDATA	ownerAuth	Authorization HMAC, key: ownerAuth.

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CMK_ApproveMA
4	20	3S	20	TPM_HMAC	outData	HMAC of migrationAuthorityDigest
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	resAuth	Authorization HMAC, key: ownerAuth.

**Action**

The TPM SHALL perform the following:

1. Validate the AuthData to use the TPM by the TPM Owner
2. Create M2 a TPM\_CMK\_MA\_APPROVAL structure

- 657       a. Set M2 ->migrationAuthorityDigest to migrationAuthorityDigest
- 658    3. Set outData = HMAC(M2) using tpmProof as the secret
- 659    4. Return TPM\_SUCCESS

## 11.7 TPM\_CMK\_CreateKey

### Start of informative comment:

The TPM\_CMK\_CreateKey command both generates and creates a secure storage bundle for asymmetric keys whose migration is controlled by a migration authority.

TPM\_CMK\_CreateKey is very similar to TPM\_CreateWrapKey, but: (1) the resultant key must be a migratable key and can be migrated only by TPM\_CMK\_CreateBlob; (2) the command is Owner authorized via a ticket.

TPM\_CMK\_CreateKey creates an otherwise normal migratable key except that (1) migrationAuth is an HMAC of the migration authority and the new key's public key, signed by tpmProof (instead of being tpmProof); (2) the migrationAuthority bit is set TRUE; (3) the payload type is TPM\_PT\_MIGRATE\_RESTRICTED.

The migration-selection/migration authority is specified by passing in a public key (actually the digests of one or more public keys, so more than one migration authority can be specified).

### End of informative comment.

## Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CMK_CreateKey
4	4			TPM_KEY_HANDLE	parentHandle	Handle of a loaded key that can perform key wrapping.
5	20	2S	20	TPM_ENCAUTH	dataUsageAuth	Encrypted usage AuthData for the sealed data.
6	◇	3S	◇	TPM_KEY12	keyInfo	Information about key to be created, pubkey.keyLength and keyInfo.encData elements are 0. MUST be TPM_KEY12
7	20	4S	20	TPM_HMAC	migrationAuthorityApproval	A ticket, created by the TPM Owner using TPM_CMK_ApproveMA, approving a TPM_MSA_COMPOSITE structure
8	20	5S	20	TPM_DIGEST	migrationAuthorityDigest	The digest of a TPM_MSA_COMPOSITE structure
9	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for parent key authorization. Must be an OSAP session.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
10	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
11	1	4H1	1	BOOL	continueAuthSession	Ignored
12	20			TPM_AUTHDATA	pubAuth	The authorization session digest that authorizes the use of the public key in parentHandle. HMAC key: parentKey.usageAuth.

## 676 Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CMK_CreateKey
4	↔	3S	↔	TPM_KEY12	wrappedKey	The TPM_KEY structure which includes the public and encrypted private key. MUST be TPM_KEY12
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, fixed at FALSE
7	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: parentKey.usageAuth.

## 677 Actions

678 The TPM SHALL do the following:

- 679 1. Validate the AuthData to use the key pointed to by parentHandle. Return  
680 TPM\_AUTHFAIL on any error
- 681 2. Validate the session type for parentHandle is OSAP
- 682 3. If the TPM is not designed to create a key of the type requested in keyInfo, return the  
683 error code TPM\_BAD\_KEY\_PROPERTY
- 684 4. Verify that parentHandle->keyUsage equals TPM\_KEY\_STORAGE
- 685 5. Verify that parentHandle-> keyFlags-> migratable == FALSE and parentHandle->  
686 encData -> migrationAuth == tpmProof
- 687 6. If keyInfo -> keyFlags -> migratable is FALSE, return TPM\_INVALID\_KEYUSAGE
- 688 7. If keyInfo -> keyFlags -> migrateAuthority is FALSE , return TPM\_INVALID\_KEYUSAGE
- 689 8. Verify that the migration authority is authorized
  - 690 a. Create M1 a TPM\_CMK\_MA\_APPROVAL structure
  - 691 i. Set M1 ->migrationAuthorityDigest to migrationAuthorityDigest
  - 692 b. Verify that migrationAuthorityApproval == HMAC(M1) using tpmProof as the secret  
693 and return error TPM\_MA\_AUTHORITY on mismatch
- 694 9. Validate key parameters
  - 695 a. keyInfo -> keyUsage MUST NOT be TPM\_KEY\_IDENTITY or  
696 TPM\_KEY\_AUTHCHANGE. If it is, return TPM\_INVALID\_KEYUSAGE
- 697 10.If TPM\_PERMANENT\_FLAGS -> FIPS is TRUE then
  - 698 a. If keyInfo -> keySize is less than 1024 return TPM\_NOTFIPS
  - 699 b. If keyInfo -> authDataUsage specifies TPM\_AUTH\_NEVER return TPM\_NOTFIPS



700       c. If keyInfo -> keyUsage specifies TPM\_KEY\_LEGACY return TPM\_NOTFIPS  
701 11.If keyInfo -> keyUsage equals TPM\_KEY\_STORAGE or TPM\_KEY\_MIGRATE  
702       a. algorithmID MUST be TPM\_ALG\_RSA  
703       b. encScheme MUST be TPM\_ES\_RSAESOAEP\_SHA1\_MGF1  
704       c. sigScheme MUST be TPM\_SS\_NONE  
705       d. key size MUST be 2048  
706 12.If keyinfo -> tag is NOT TPM\_TAG\_KEY12 return error TPM\_INVALID\_STRUCTURE  
707 13.Map wrappedKey to a TPM\_KEY12 structure  
708 14.If authHandle indicates XOR encryption for the AuthData secrets  
709       a. Create X1 the SHA-1 of the concatenation of (authHandle -> sharedSecret ||  
710       authLastNonceEven)  
711       b. Create DU1 by XOR X1 and dataUsageAuth  
712 15.Else  
713       a. Create DU1 by decrypting dataUsageAuth using the algorithm indicated in the OSAP  
714       session  
715       b. Key is from authHandle -> sharedSecret  
716       c. IV is SHA-1 of (authLastNonceEven || nonceOdd)  
717 16.Set continueAuthSession to FALSE  
718 17.Generate asymmetric key according to algorithm information in keyInfo  
719 18.Fill in the wrappedKey structure with information from the newly generated key.  
720       a. Set wrappedKey -> encData -> usageAuth to DU1  
721       b. Set wrappedKey -> encData -> payload to TPM\_PT\_MIGRATE\_RESTRICTED  
722       c. Create thisPubKey, a TPM\_PUBKEY structure containing wrappedKey's public key  
723       and algorithm parameters  
724       d. Create M2 a TPM\_CMK\_MIGAUTH structure  
725           i. Set M2 -> msaDigest to migrationAuthorityDigest  
726           ii. Set M2 -> pubKeyDigest to SHA-1 (thisPubKey)  
727       e. Set wrappedKey -> encData -> migrationAuth equal to HMAC(M2), using tpmProof as  
728       the shared secret  
729 19.If wrappedKey->PCRInfoSize is non-zero  
730       a. Set wrappedKey -> pcrInfo to a TPM\_PCR\_INFO\_LONG structure  
731       b. Set digestAtCreation to the TPM\_COMPOSITE\_HASH indicated by  
732       creationPCRSelection  
733       c. Set localityAtCreation to TPM\_STANY\_FLAGS -> localityModifier  
734 20.Encrypt the private portions of the wrappedKey structure using the key in parentHandle  
735 21.Return the newly generated key in the wrappedKey parameter

## 11.8 TPM\_CMK\_CreateTicket

### Start of informative comment:

The TPM\_CMK\_CreateTicket command uses a public key to verify the signature over a digest.

TPM\_CMK\_CreateTicket returns a ticket that can be used to prove to the same TPM that signature verification with a particular public key was successful.

### End of informative comment.

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CMK_CreateTicket
4	<>	2S	<>	TPM_PUBKEY	verificationKey	The public key to be used to check signatureValue
5	20	3S	20	TPM_DIGEST	signedData	The data to be verified
6	4	4S	4	UINT32	signatureValueSize	The size of the signatureValue
7	<>	5S	<>	BYTE[]	signatureValue	The signatureValue to be verified
8	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
9	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
10	1	4H1	1	BOOL	continueAuthSession	Ignored
11	20			TPM_AUTHDATA	pubAuth	The authorization session digest for inputs and owner. HMAC key: ownerAuth.

744 **Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CMK_CreateTicket
4	20	3S	20	TPM_HMAC	sigTicket	Ticket that proves digest created on this TPM
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag
7	20			TPM_AUTHDATA	resAuth	Authorization. HMAC key: ownerAuth.

745 **Actions**

746 The TPM SHALL do the following:

- 747 1. Validate the TPM Owner authentication to use the command
- 748 2. Validate that the key type and algorithm are correct
- 749     a. Validate that verificationKey -> algorithmParms -> algorithmID == TPM\_ALG\_RSA
- 750     b. Validate that verificationKey -> algorithmParms -> encScheme == TPM\_ES\_NONE
- 751     c. Validate that verificationKey -> algorithmParms -> sigScheme is
- 752     TPM\_SS\_RSASSAPKCS1v15\_SHA1
- 753 3. Use verificationKey to verify that signatureValue is a valid signature on signedData, and
- 754     return error TPM\_BAD\_SIGNATURE on mismatch
- 755 4. Create M2 a TPM\_CMK\_SIGTICKET
- 756     a. Set M2 -> verKeyDigest to the SHA-1 (verificationKey)
- 757     b. Set M2 -> signedData to signedData
- 758 5. Set sigTicket = HMAC(M2) signed by using tpmProof as the secret
- 759 6. Return TPM\_SUCCESS

## 11.9 TPM\_CMK\_CreateBlob

### **Start of informative comment:**

TPM\_CMK\_CreateBlob command is very similar to TPM\_CreateMigrationBlob, except that it:  
(1) uses an extra ticket (restrictedKeyAuth) instead of a migrationAuth authorization session; (2) uses the migration options TPM\_MS\_RESTRICT\_MIGRATE or TPM\_MS\_RESTRICT\_APPROVE\_DOUBLE; (3) produces a wrapped key blob whose migrationAuth is independent of tpmProof.

If the destination (parent) public key is the MA, migration is implicitly permitted. Further checks are required if the MA is not the destination (parent) public key, and merely selects a migration destination: (1) sigTicket must prove that restrictTicket was signed by the MA; (2) restrictTicket must vouch that the target public key is approved for migration to the destination (parent) public key. (Obviously, this more complex method may also be used by an MA to approve migration to that MA.) In both cases, the MA must be one of the MAs implicitly listed in the migrationAuth of the target key-to-be-migrated.

### **End of informative comment.**

## 775 Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CMK_CreateBlob
4	4			TPM_KEY_HANDLE	parentHandle	Handle of the parent key that can decrypt encData.
5	2	2S	2	TPM_MIGRATE_SCHEME	migrationType	The migration type, either TPM_MS_RESTRICT_MIGRATE or TPM_MS_RESTRICT_APPROVE_DOUBLE
6	<>	3S	<>	TPM_MIGRATIONKEYAUTH	migrationKeyAuth	Migration public key and its authorization session digest.
7	20	4S	20	TPM_DIGEST	pubSourceKeyDigest	The digest of the TPM_PUBKEY of the entity to be migrated
8	4	5S	4	UINT32	msaListSize	The size of the msaList parameter, which is a variable length TPM_MSA_COMPOSITE structure
9	<>	6S	<>	TPM_MSA_COMPOSITE	msaList	One or more digests of public keys belonging to migration authorities
10	4	7S	4	UINT32	restrictTicketSize	The size of the restrictTicket parameter, which is a TPM_CMK_AUTH structure if migration type is TPM_MS_RESTRICT_APPROVE_DOUBLE
11	<>	8S	<>	BYTE[]	restrictTicket	Either a NULL parameter or a TPM_CMK_AUTH structure, containing the digests of the public keys belonging to the Migration Authority, the destination parent key and the key-to-be-migrated.
12	4	9S	4	UINT32	sigTicketSize	The size of the sigTicket parameter, which is a TPM_HMAC structure if migration type is TPM_MS_RESTRICT_APPROVE_DOUBLE.
13	<>	10S	<>	BYTE[]	sigTicket	Either a NULL parameter or a TPM_HMAC structure, generated by the TPM, signaling a valid signature over restrictTicket
14	4	11S	4	UINT32	encDataSize	The size of the encData parameter
15	<>	12S	<>	BYTE[]	encData	The encrypted entity that is to be modified.
16	4			TPM_AUTHHANDLE	parentAuthHandle	The authorization session handle used for the parent key.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
17	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with parentAuthHandle
18	1	4H1	1	BOOL	continueAuthSession	Continue use flag for parent session
19	20		20	TPM_AUTHDATA	parentAuth	HMAC key: parentKey.usageAuth.

## 776 Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CMK_CreateBlob
4	4	3S	4	UINT32	randomSize	The used size of the output area for random
5	<>	4S	<>	BYTE[]	random	String used for xor encryption
6	4	5S	4	UINT32	outDataSize	The used size of the output area for outData
7	<>	6S	<>	BYTE[]	outData	The modified, encrypted entity.
8	20	3H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		4H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with parentAuthHandle
9	1	5H1	1	BOOL	continueAuthSession	Continue use flag for parent key session
10	20		20	TPM_AUTHDATA	resAuth	HMAC key: parentKey.usageAuth.

## 777 Description

778 The TPM does not check the PCR values when migrating values locked to a PCR.

## 779 Actions

- 780 1. Validate that parentAuth authorizes the use of the key pointed to by parentHandle.
- 781 2. Verify that parentHandle-> keyFlags-> migratable == FALSE and parentHandle->  
782 encData -> migrationAuth == tpmProof
- 783 3. Create d1 by decrypting encData using the key pointed to by parentHandle.
- 784 4. Verify that the digest within migrationKeyAuth is legal for this TPM and public key
- 785 5. Verify that d1 -> payload == TPM\_PT\_MIGRATE\_RESTRICTED or  
786 TPM\_PT\_MIGRATE\_EXTERNAL
- 787 6. Verify that the migration authorities in msaList are authorized to migrate this key
  - 788 a. Create M2 a TPM\_CMK\_MIGAUTH structure
    - 789 i. Set M2 -> msaDigest to SHA1[msaList]
    - 790 ii. Set M2 -> pubKeyDigest to pubSourceKeyDigest
  - 791 b. Verify that d1 -> migrationAuth == HMAC(M2) using tpmProof as the secret and  
792 return error TPM\_MA\_AUTHORITY on mismatch
- 793 7. If migrationKeyAuth -> migrationScheme == TPM\_MS\_RESTRICT\_MIGRATE
  - 794 a. Verify that intended migration destination is an MA:
    - 795 i. For one of n=1 to n=(msaList -> MSAList), verify that SHA1[migrationKeyAuth ->  
796 migrationKey] == msaList -> migAuthDigest[n]
  - 797 b. Validate that the MA key is the correct type

```

798         i. Validate that migrationKeyAuth -> migrationKey -> algorithmParms ->
799           algorithmID == TPM_ALG_RSA
800         ii. Validate that migrationKeyAuth -> migrationKey -> algorithmParms -> encScheme
801            is an encryption scheme supported by the TPM
802         iii. Validate that migrationKeyAuth -> migrationKey -> algorithmParms -> sigScheme
803            is TPM_SS_NONE
804     8. else If migrationKeyAuth -> migrationScheme ==
805        TPM_MS_RESTRICT_APPROVE_DOUBLE,
806     a. Verify that the intended migration destination has been approved by the MSA:
807         i. Verify that for one of the n=1 to n=(msaList -> MSAList) values of msaList ->
808            migAuthDigest[n], sigTicket == HMAC (V1) using tpmProof as the secret where V1
809            is a TPM_CMK_SIGTICKET structure such that:
810            (1) V1 -> verKeyDigest = msaList -> migAuthDigest[n]
811            (2) V1 -> signedData = SHA1[restrictTicket]
812         ii. If [restrictTicket -> destinationKeyDigest] != SHA1[migrationKeyAuth ->
813            migrationKey], return error TPM_MA_DESTINATION
814         iii. If [restrictTicket -> sourceKeyDigest] != pubSourceKeyDigest, return error
815            TPM_MA_SOURCE
816     9. Else return with error TPM_BAD_PARAMETER.
817     10. Build two bytes array, K1 and K2, using d1:
818         a. K1 = TPM_STORE_ASYMKEY.privKey[0..19]
819            (TPM_STORE_ASYMKEY.privKey.keyLength + 16 bytes of
820            TPM_STORE_ASYMKEY.privKey.key), sizeof(K1) = 20
821         b. K2 = TPM_STORE_ASYMKEY.privKey[20..131] (position 16-127 of
822            TPM_STORE_ASYMKEY . privKey.key), sizeof(K2) = 112
823     11. Build M1 a TPM_MIGRATE_ASYMKEY structure
824         a. TPM_MIGRATE_ASYMKEY.payload = TPM_PT_CMK_MIGRATE
825         b. TPM_MIGRATE_ASYMKEY.usageAuth = TPM_STORE_ASYMKEY.usageAuth
826         c. TPM_MIGRATE_ASYMKEY.pubDataDigest = TPM_STORE_ASYMKEY. pubDataDigest
827         d. TPM_MIGRATE_ASYMKEY.partPrivKeyLen = 112 - 127.
828         e. TPM_MIGRATE_ASYMKEY.partPrivKey = K2
829     12. Create o1 (which SHALL be 198 bytes for a 2048 bit RSA key) by performing the OAEP
830        encoding of m using OAEP parameters m, pHash, and seed
831         a. m is the previously created M1
832         b. pHash = SHA1( SHA1[msaList] || pubSourceKeyDigest)
833         c. seed = s1 = the previously created K1
834     13. Create r1 a random value from the TPM RNG. The size of r1 MUST be the size of o1.
835        Return r1 in the random parameter

```

- 836 14. Create x1 by XOR of o1 with r1
- 837 15. Copy r1 into the output field "random"
- 838 16. Encrypt x1 with the migrationKeyAuth-> migrationKey



**11.10TPM\_CMK\_ConvertMigration****Start of informative comment:**

TPM\_CMK\_ConvertMigration completes the migration of certified migration blobs.

This command takes a certified migration blob and creates a normal wrapped blob with payload type TPM\_PT\_MIGRATE\_EXTERNAL. The migrated blob must be loaded into the TPM using the normal TPM\_LoadKey function.

Note that the command migrates private keys, only. The migration of the associated public keys is not specified by TPM because they are not security sensitive. Migration of the associated public keys may be specified in a platform specific specification. A TPM\_KEY structure must be recreated before the migrated key can be used by the target TPM in a TPM\_LoadKey command.

TPM\_CMK\_ConvertMigration checks that one of the MAs implicitly listed in the migrationAuth of the target key has approved migration of the target key to the destination (parent) key, and that the settings (flags etc.) in the target key are those of a CMK.

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CMK_ConvertMigration
4	4			TPM_KEY_HANDLE	parentHandle	Handle of a loaded key that can decrypt keys.
5	60	2S	60	TPM_CMK_AUTH	restrictTicket	The digests of public keys belonging to the Migration Authority, the destination parent key and the key -to-be-migrated.
6	20	3S	20	TPM_HMAC	sigTicket	A signature ticket, generated by the TPM, signaling a valid signature over restrictTicket
7	◇	4S	◇	TPM_KEY12	migratedKey	The public key of the key -to-be-migrated. The private portion MUST be TPM_MIGRATE_ASYMKEY properly XOR'd
8	4	5S	4	UINT32	msaListSize	The size of the msaList parameter, which is a variable length TPM_MSA_COMPOSITEstructure
9	◇	6S	◇	TPM_MSA_COMPOSITE	msaList	One or more digests of public keys belonging to migration authorities
10	4	7S	4	UINT32	randomSize	Size of random
11	◇	8S	◇	BYTE [ ]	random	Random value used to hide key data.
12	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
13	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
14	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
15	20			TPM_AUTHDATA	parentAuth	Authorization HMAC : parentKey.usageAuth

## 855 Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CMK_ConvertMigration
4	4	3S	4	UINT32	outDataSize	The used size of the output area for outData
5	◇	4S	◇	BYTE[]	outData	The encrypted private key that can be loaded with TPM_LoadKey
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20			TPM_AUTHDATA	resAuth	Authorization HMAC key .usageAuth

## 856 Action

- 857 1. Validate the AuthData to use the key in parentHandle
- 858 2. If the keyUsage field of the key referenced by parentHandle does not have the value
- 859 TPM\_KEY\_STORAGE, the TPM must return the error code TPM\_INVALID\_KEYUSAGE
- 860 3. Create d1 by decrypting the migratedKey -> encData area using the key in parentHandle
- 861 4. Create o1 by XOR d1 and random parameter
- 862 5. Create m1 a TPM\_MIGRATE\_ASYMKEY, seed and pHash by OAEP decoding o1
- 863 6. Create migratedPubKey a TPM\_PUBKEY structure corresponding to migratedKey
- 864 a. Verify that pHash == SHA1( SHA1[msaList] || SHA1(migratedPubKey )
- 865 7. Create k1 by combining seed and the TPM\_MIGRATE\_ASYMKEY -> partPrivKey field
- 866 8. Create d2 a TPM\_STORE\_ASYMKEY structure.
- 867 a. Set the TPM\_STORE\_ASYMKEY -> privKey field to k1
- 868 b. Set d2 -> usageAuth to m1 -> usageAuth
- 869 c. Set d2 -> pubDataDigest to m1 -> pubDataDigest
- 870 9. Verify that parentHandle-> keyFlags -> migratable == FALSE and parentHandle->
- 871 encData -> migrationAuth == tpmProof
- 872 10. Verify that m1 -> payload == TPM\_PT\_CMK\_MIGRATE then set d2-> payload =
- 873 TPM\_PT\_MIGRATE\_EXTERNAL
- 874 11. Verify that for one of the n=1 to n=(msaList -> MSAList) values of msaList ->
- 875 migAuthDigest[n] sigTicket == HMAC (V1) using tpmProof as the secret where V1 is a
- 876 TPM\_CMK\_SIGTICKET structure such that:
- 877 a. V1 -> verKeyDigest = msaList -> migAuthDigest[n]
- 878 b. V1 -> signedData = SHA1[restrictTicket]

879 12.Create parentPubKey, a TPM\_PUBKEY structure corresponding to parentHandle  
880 13.If [restrictTicket -> destinationKeyDigest] != SHA1(parentPubKey), return error  
881 TPM\_MA\_DESTINATION  
882 14.Verify that migratedKey is corresponding to d2  
883 15.If migratedKey -> keyFlags -> migratable is FALSE, and return error  
884 TPM\_INVALID\_KEYUSAGE  
885 16.If migratedKey -> keyFlags -> migrateAuthority is FALSE, return error  
886 TPM\_INVALID\_KEYUSAGE  
887 17.If [restrictTicket -> sourceKeyDigest] != SHA1(migratedPubKey), return error  
888 TPM\_MA\_SOURCE  
889 18.Create M2 a TPM\_CMK\_MIGAUTH structure  
890 a. Set M2 -> msaDigest to SHA1[msaList]  
891 b. Set M2 -> pubKeyDigest to SHA1[migratedPubKey]  
892 19.Set d2 -> migrationAuth = HMAC(M2) using tpmProof as the secret  
893 20.Create outData using the key in parentHandle to perform the encryption

## 12. Maintenance Functions (optional)

### **Start of informative comment:**

When a maintenance archive is created with generateRandom FALSE, the maintenance blob is XOR encrypted with the owner authorization before encryption with the maintenance public key. This prevents the manufacturer from obtaining plaintext data. The receiving TPM must have the same owner authorization as the sending TPM in order to XOR decrypt the archive.

When generateRandom is TRUE, the maintenance blob is XOR encrypted with random data, which is also returned. This permits someone trusted by the Owner to load the maintenance archive into the replacement platform in the absence of the Owner and manufacturer, without the Owner having to reveal information about his auth value. The receiving and sending TPM's may have different owner authorizations. The random data is transferred from the sending TPM owner to the receiving TPM owner out of band, so the maintenance blob remains hidden from the manufacturer.

This is a typical maintenance sequence:

#### 1. Manufacturer:

- generates maintenance key pair
- gives public key to TPM1 owner

#### 2. TPM1: TPM\_LoadManuMaintPub

- load maintenance public key

#### 3. TPM1: TPM\_CreateMaintenanceArchive

- XOR encrypt with owner auth or random
- encrypt with maintenance public key

#### 4. Manufacturer:

- decrypt with maintenance private key
- (still XOR encrypted with owner auth or random)

- encrypt with TPM2 SRK public key

#### 5. TPM2: TPM\_LoadMaintenanceArchive

- decrypt with SRK private key
- XOR decrypt with owner auth or random

### **End of informative comment.**

## 12.1 TPM\_CreateMaintenanceArchive

**Start of informative comment:**

This command creates the maintenance archive. It can only be executed by the owner, and may be shut off with the TPM\_KillMaintenanceFeature command.

**End of informative comment.**

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Cmd ordinal: TPM_ORD_CreateMaintenanceArchive
4	1	2S	1	BOOL	generateRandom	Use RNG or Owner auth to generate 'random'.
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
8	20			TPM_AUTHDATA	ownerAuth	HMAC key: ownerAuth.

### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Cmd ordinal: TPM_ORD_CreateMaintenanceArchive
4	4	3S	4	UINT32	randomSize	Size of the returned random data. Will be 0 if generateRandom is FALSE.
5	◇	4S	◇	BYTE []	random	Random data to XOR with result.
6	4	5S	4	UINT32	archiveSize	Size of the encrypted archive
7	◇	6S	◇	BYTE []	archive	Encrypted key archive.
8	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
10	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

### Actions

Upon authorization being confirmed this command does the following:

- 936 1. Validates that the TPM\_PERMANENT\_FLAGS -> allowMaintenance is TRUE. If it is  
937 FALSE, the TPM SHALL return TPM\_DISABLED\_CMD and exit this capability.
- 938 2. Validates the TPM Owner AuthData.
- 939 3. If the value of TPM\_PERMANENT\_DATA -> manuMaintPub is zero, the TPM MUST  
940 return the error code TPM\_KEYNOTFOUND
- 941 4. Build a1 a TPM\_KEY structure using the SRK. The encData field is not a normal  
942 TPM\_STORE\_ASYMKEY structure but rather a TPM\_MIGRATE\_ASYMKEY structure built  
943 using the following actions.
- 944 5. Build a TPM\_STORE\_PRIVKEY structure from the SRK. This privKey element should be  
945 132 bytes long for a 2K RSA key.
- 946 6. Create k1 and k2 by splitting the privKey element created in step 4 into 2 parts. k1 is  
947 the first 20 bytes of privKey, k2 contains the remainder of privKey.
- 948 7. Build m1 by creating and filling in a TPM\_MIGRATE\_ASYMKEY structure
- 949 a. m1 -> usageAuth is set to TPM\_PERMANENT\_DATA -> tpmProof
- 950 b. m1 -> pubDataDigest is set to the digest value of the SRK fields from step 4
- 951 c. m1 -> payload is set to TPM\_PT\_MAINT
- 952 d. m1 -> partPrivKey is set to k2
- 953 8. Create o1 (which SHALL be 198 bytes for a 2048 bit RSA key) by performing the OAEP  
954 encoding of m using OAEP parameters of
- 955 a. m = TPM\_MIGRATE\_ASYMKEY structure (step 7)
- 956 b. pHash = TPM\_PERMANENT\_DATA -> ownerAuth
- 957 c. seed = s1 = k1 (step 6)
- 958 9. If generateRandom = TRUE
- 959 a. Create r1 by obtaining values from the TPM RNG. The size of r1 MUST be the same  
960 size as o1. Set random parameter to r1
- 961 10.If generateRandom = FALSE
- 962 a. Create r1 by applying MGF1 to the TPM Owner AuthData. The size of r1 MUST be the  
963 same size as o1. Set random parameter to null.
- 964 11.Create x1 by XOR of o1 with r1
- 965 12.Encrypt x1 with the manuMaintPub key using the TPM\_ES\_RSAESOAEP\_SHA1\_MGF1  
966 encryption scheme.
- 967 13.Set a1 -> encData to the encryption of x1
- 968 14.Set TPM\_PERMANENT\_FLAGS -> maintenanceDone to TRUE
- 969 15.Return a1 in the archive parameter

**12.2 TPM\_LoadMaintenanceArchive****Start of informative comment:**

This command loads in a Maintenance archive that has been massaged by the manufacturer to load into another TPM.

If the maintenance archive was created using the owner authorization for XOR encryption, the current owner authorization must be used for decryption. The owner authorization does not change.

If the maintenance archive was created using random data for the XOR encryption, the vendor specific arguments must include the random data. The owner authorization may change.

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadMaintenanceArchive
4	4	2S	4	UINT32	archiveSize	Size of the encrypted archive
5	◇	3S	◇	BYTE[]	archive	Encrypted key archive
				...	...	Vendor specific arguments
-	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
			20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
-	20		20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
-	1		1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
--	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner authentication. HMAC key: ownerAuth.

## 983 Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4		4	TPM_RESULT	returnCode	The return code of the operation.
			4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadMaintenanceArchive
				..	..	Vendor specific arguments
-	20		20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
			20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
-	1		1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
-	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth, the original value and not the new auth value

## 984 Descriptions

985 The maintenance mechanisms in the TPM MUST not require the TPM to hold a global  
986 secret. The definition of global secret is a secret value shared by more than one TPM.

987 The TPME is not allowed to pre-store or use unique identifiers in the TPM for the purpose of  
988 maintenance. The TPM MUST NOT use the endorsement key for identification or encryption  
989 in the maintenance process. The maintenance process MAY use a TPM Identity to deliver  
990 maintenance information to specific TPM's.

991 The maintenance process can only change the SRK, tpmProof and TPM Owner AuthData  
992 fields.

993 The maintenance process can only access data in shielded locations where this data is  
994 necessary to validate the TPM Owner, validate the TPME and manipulate the blob

995 The TPM MUST be conformant to the TPM specification, protection profiles and security  
996 targets after maintenance. The maintenance MAY NOT decrease the security values from  
997 the original security target.

998 The security target used to evaluate this TPM MUST include this command in the TOE.

## 999 Actions

000 The TPM SHALL perform the following when executing the command

- 001 1. Validate the TPM Owner's AuthData
- 002 2. Validate that the maintenance information was sent by the TPME. The validation  
003 mechanism MUST use a strength of function that is at least the same strength of  
004 function as a digital signature performed using a 2048 bit RSA key.
- 005 3. The packet MUST contain m2 as defined in section 12.1.
- 006 4. Ensure that only the target TPM can interpret the maintenance packet. The protection  
007 mechanism MUST use a strength of function that is at least the same strength of  
008 function as a digital signature performed using a 2048 bit RSA key.
- 009 5. Process the maintenance information



- 010      a. Update the SRK
- 011          i. Set the SRK usageAuth to be the same as the source TPM owner's AuthData
- 012      b. Update TPM\_PERMANENT\_DATA -> tpmProof
- 013      c. Update TPM\_PERMANENT\_DATA -> ownerAuth
- 014    6. Set TPM\_PERMANENT\_FLAGS -> maintenanceDone to TRUE
- 015    7. Terminate all OSAP and DSAP sessions

## 12.3 TPM\_KillMaintenanceFeature

### Informative Comments:

The TPM\_KillMaintenanceFeature is a permanent action that prevents ANYONE from creating a maintenance archive. This action, once taken, is permanent until a new TPM Owner is set.

This action is to allow those customers who do not want the maintenance feature to not allow the use of the maintenance feature.

At the discretion of the Owner, it should be possible to kill the maintenance feature in such a way that the only way to recover maintainability of the platform would be to wipe out the root keys. This feature is mandatory in any TPM that implements the maintenance feature.

### End informative Comment

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_KillMaintenanceFeature
4	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
5	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
7	20			TPM_AUTHDATA	ownerAuth	HMAC key: ownerAuth.

### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_KillMaintenanceFeature
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	HMAC key: ownerAuth.

### Actions

1. Validate the TPM Owner AuthData
2. Set the TPM\_PERMANENT\_FLAGS.allowMaintenance flag to FALSE.

## 12.4 TPM\_LoadManuMaintPub

### Informative Comments:

The TPM\_LoadManuMaintPub command loads the manufacturer's public key for use in the maintenance process. The command installs manuMaintPub in PERMANENT data storage inside a TPM. Maintenance enables duplication of non-migratory data in protected storage. There is therefore a security hole if a platform is shipped before the maintenance public key has been installed in a TPM.

The command is expected to be used before installation of a TPM Owner or any key in TPM protected storage. It therefore does not use authorization.

### End of Informative Comments

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadManuMaintPub
4	20	2S	20	TPM_NONCE	antiReplay	AntiReplay and validation nonce
5	<	3S	<	TPM_PUBKEY	pubKey	The public key of the manufacturer to be in use for maintenance

### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadManuMaintPub
4	20	3S	20	TPM_DIGEST	checksum	Digest of pubKey and antiReplay

### Description

The pubKey MUST specify an algorithm whose strength is not less than the RSA algorithm with 2048bit keys.

pubKey SHOULD unambiguously identify the entity that will perform the maintenance process with the TPM Owner.

TPM\_PERMANENT\_DATA -> manuMaintPub SHALL exist in a TPM-shielded location, only.

If an entity (Platform Entity) does not support the maintenance process but issues a platform credential for a platform containing a TPM that supports the maintenance process, the value of TPM\_PERMANENT\_DATA -> manuMaintPub MUST be set to zero before the platform leaves the entity's control. That is, this ordinal can only be run once, and used to either load the key or load a NULL key.

055 **Actions**

- 056 The first valid TPM\_LoadManuMaintPub command received by a TPM SHALL
- 057 1. Store the parameter pubKey as TPM\_PERMANENT\_DATA -> manuMaintPub.
- 058 2. Set checksum to SHA-1 of (pubKey || antiReplay)
- 059 3. Export the checksum
- 060 4. Subsequent calls to TPM\_LoadManuMaintPub SHALL return code
- 061 TPM\_DISABLED\_CMD.

## 12.5 TPM\_ReadManuMaintPub

### Informative Comments:

The TPM\_ReadManuMaintPub command is used to check whether the manufacturer's public maintenance key in a TPM has the expected value. This may be useful during the manufacture process. The command returns a digest of the installed key, rather than the key itself. This hinders discovery of the maintenance key, which may (or may not) be useful for manufacturer privacy.

The command is expected to be used before installation of a TPM Owner or any key in TPM protected storage. It therefore does not use authorization.

### End of Informative Comments

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReadManuMaintPub
4	20	2S	20	TPM_NONCE	antiReplay	AntiReplay and validation nonce

### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReadManuMaintPub
4	20	3S	20	TPM_DIGEST	checksum	Digest of pubKey and antiReplay

### Description

This command returns the hash of the antiReplay nonce and the previously loaded manufacturer's maintenance public key.

### Actions

The TPM\_ReadManuMaintPub command SHALL

1. Create "checksum" by concatenating data to form (TPM\_PERMANENT\_DATA -> manuMaintPub || antiReplay) and passing the concatenated data through SHA1.
2. Export the checksum

## 13. Cryptographic Functions

### 13.1 TPM\_SHA1Start

#### Start of informative comment:

This capability starts the process of calculating a SHA-1 digest.

The exposure of the SHA-1 processing is a convenience to platforms in a mode that do not have sufficient memory to perform SHA-1 themselves. As such, the use of SHA-1 is restrictive on the TPM.

The TPM may not allow any other types of processing during the execution of a SHA-1 session. There is only one SHA-1 session active on a TPM.

After the execution of TPM\_SHA1Start, and prior to TPM\_SHA1Complete or TPM\_SHA1CompleteExtend, the receipt of any command other than TPM\_SHA1Update will cause the invalidation of the SHA-1 session.

#### End of informative comment.

#### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_SHA1Start

#### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_SHA1Start
4	4	3S	4	UINT32	maxNumBytes	Maximum number of bytes that can be sent to TPM_SHA1Update. Must be a multiple of 64 bytes.

#### Description

1. This capability prepares the TPM for a subsequent TPM\_SHA1Update, TPM\_SHA1Complete or TPM\_SHA1CompleteExtend command. The capability SHALL open a thread that calculates a SHA-1 digest.
2. After receipt to TPM\_SHA1Start, and prior to the receipt of TPM\_SHA1Complete or TPM\_SHA1CompleteExtend, receipt of any command other than TPM\_SHA1Update invalidates the SHA-1 session.
  - a. If the command received is TPM\_ExecuteTransport, the SHA-1 session invalidation is based on the wrapped command, not the TPM\_ExecuteTransport ordinal.

**13.2 TPM\_SHA1Update****Start of informative comment:**

This capability inputs complete blocks of data into a pending SHA-1 digest. At the end of the process, the digest remains pending.

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_SHA1Update
4	4	2S	4	UINT32	numBytes	The number of bytes in hashData. Must be a multiple of 64 bytes.
5	∞	3S	∞	BYTE []	hashData	Bytes to be hashed

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_SHA1Update

**Description**

This command SHALL incorporate complete blocks of data into the digest of an existing SHA-1 thread. Only integral numbers of complete blocks (64 bytes each) can be processed.

### 13.3 TPM\_SHA1Complete

**Start of informative comment:**

This capability terminates a pending SHA-1 calculation.

**End of informative comment.**

#### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SHA1Complete
4	4	2S	4	UINT32	hashDataSize	Number of bytes in hashData, MUST be 64 or less
5	◇	3S	◇	BYTE []	hashData	Final bytes to be hashed

#### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SHA1Complete
4	20	3S	20	TPM_DIGEST	hashValue	The output of the SHA-1 hash.

#### Description

This command SHALL incorporate a partial or complete block of data into the digest of an existing SHA-1 thread, and terminate that thread. hashDataSize MAY have values in the range of 0 through 64, inclusive.

If the SHA-1 thread has received no bytes the TPM SHALL calculate the SHA-1 of the empty buffer.



**13.4 TPM\_SHA1CompleteExtend****Start of informative comment:**

This capability terminates a pending SHA-1 calculation and EXTENDS the result into a Platform Configuration Register using a SHA-1 hash process.

This command is designed to complete a hash sequence and extend a PCR in memory-less environments.

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SHA1CompleteExtend
4	4	2S	4	TPM_PCRINDEX	pcrNum	Index of the PCR to be modified
5	4	3S	4	UINT32	hashDataSize	Number of bytes in hashData, MUST be 64 or less
6	<>	4S	<>	BYTE []	hashData	Final bytes to be hashed

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SHA1CompleteExtend
4	20	3S	20	TPM_DIGEST	hashValue	The output of the SHA-1 hash.
5	20	4S	20	TPM_PCRVALUE	outDigest	The PCR value after execution of the command.

**Description**

This command SHALL incorporate a partial or complete block of data into the digest of an existing SHA-1 thread, EXTEND the resultant digest into a PCR, and terminate the SHA-1 session. hashDataSize MAY have values in the range of 0 through 64, inclusive.

The SHA-1 session MUST terminate even if the command returns an error, e.g. TPM\_BAD\_LOCALITY.

**Actions**

1. Map V1 to TPM\_STANY\_DATA
2. Map L1 to V1 -> localityModifier
3. If the current locality, held in L1, is not selected in TPM\_PERMANENT\_DATA -> pcrAttrib [pcrNum]. pcrExtendLocal, return TPM\_BAD\_LOCALITY

- 148 4. Create H1 the TPM\_DIGEST of the SHA-1 session ensuring that hashData, if any, is  
149 added to the SHA-1 session
- 150 5. Perform the actions of TPM\_Extend using H1 as the data and pcrNum as the PCR to  
151 extend

**13.5 TPM\_Sign****Start of informative comment:**

The Sign command signs data and returns the resulting digital signature

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Sign.
4	4			TPM_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key that can perform digital signatures.
5	4	2s	4	UINT32	areaToSignSize	The size of the areaToSign parameter
6	<>	3s	<>	BYTE[]	areaToSign	The value to sign
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
10	20			TPM_AUTHDATA	privAuth	The authorization session digest that authorizes the use of keyHandle. HMAC key: key.usageAuth

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Sign.
4	4	3S	4	UINT32	sigSize	The length of the returned digital signature
5	<>	4S	<>	BYTE[]	sig	The resulting digital signature.
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: key.usageAuth

**Description**

The TPM MUST support all values of areaToSignSize that are legal for the defined signature scheme and key size. The maximum value of areaToSignSize is determined by the defined signature scheme and key size.

162 In the case of PKCS1v15\_SHA1 the areaToSignSize MUST be TPM\_DIGEST (the hash size of  
163 a SHA-1 operation - see 8.5.1 TPM\_SS\_RSASSAPKCS1v15\_SHA1). In the case of  
164 PKCS1v15\_DER the maximum size of areaToSign is k-11 octets, where k is limited by the  
165 key size (see TPM\_SS\_RSASSAPKCS1v15\_DER).

## 166 **Actions**

- 167 1. The TPM validates the AuthData to use the key pointed to by keyHandle.
- 168 2. If the areaToSignSize is 0 the TPM returns TPM\_BAD\_PARAMETER.
- 169 3. Validate that keyHandle -> keyUsage is TPM\_KEY\_SIGNING or TPM\_KEY\_LEGACY, if not  
170 return the error code TPM\_INVALID\_KEYUSAGE
- 171 4. The TPM verifies that the signature scheme and key size can properly sign the  
172 areaToSign parameter.
- 173 5. If signature scheme is TPM\_SS\_RSASSAPKCS1v15\_SHA1 then  
174 a. Validate that areaToSignSize is 20 return TPM\_BAD\_PARAMETER on error  
175 b. Set S1 to areaToSign
- 176 6. Else if signature scheme is TPM\_SS\_RSASSAPKCS1v15\_DER then  
177 a. Validate that areaToSignSize is at least 11 bytes less than the key size, return  
178 TPM\_BAD\_PARAMETER on error  
179 b. Set S1 to areaToSign
- 180 7. else if signature scheme is TPM\_SS\_RSASSAPKCS1v15\_INFO then  
181 a. Create S2 a TPM\_SIGN\_INFO structure  
182 b. Set S2 -> fixed to "SIGN"  
183 c. Set S2 -> replay to nonceOdd  
184 i. If nonceOdd is not present due to an unauthorized command return  
185 TPM\_BAD\_PARAMETER  
186 d. Set S2 -> dataLen to areaToSignSize  
187 e. Set S2 -> data to areaToSign  
188 f. Set S1 to the SHA-1(S2)
- 189 8. Else return TPM\_INVALID\_KEYUSAGE
- 190 9. The TPM computes the signature, sig, using the key referenced by keyHandle using S1  
191 as the value to sign
- 192 10. Return the computed signature in Sig

**13.6 TPM\_GetRandom****Start of informative comment:**

TPM\_GetRandom returns the next bytesRequested bytes from the random number generator to the caller.

It is recommended that a TPM implement the RNG in a manner that would allow it to return RNG bytes such that the frequency of bytesRequested being more than the number of bytes available is an infrequent occurrence.

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetRandom.
4	4	2S	4	UINT32	bytesRequested	Number of bytes to return

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetRandom.
4	4	3S	4	UINT32	randomBytesSize	The number of bytes returned
5	◊	4S	◊	BYTE[]	randomBytes	The returned bytes

**Actions**

1. The TPM determines if amount bytesRequested is available from the TPM.
2. Set randomBytesSize to the number of bytes available from the RNG. This number MAY be less than bytesRequested
3. Set randomBytes to the next randomBytesSize bytes from the RNG

## 13.7 TPM\_StirRandom

**Start of informative comment:**

TPM\_StirRandom adds entropy to the RNG state.

**End of informative comment.**

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_StirRandom
4	4	2S	4	UINT32	dataSize	Number of bytes of input (<256)
5	◇	3S	◇	BYTE[]	inData	Data to add entropy to RNG state

### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_StirRandom

### Actions

The TPM updates the state of the current RNG using the appropriate mixing function.

## 13.8 TPM\_CertifyKey

**Start of informative comment:**

The TPM\_CertifyKey operation allows one key to certify the public portion of another key.

A TPM identity key may be used to certify non-migratable keys but is not permitted to certify migratory keys or certified migration keys. As such, it allows the TPM to make the statement “this key is held in a TPM-shielded location, and it will never be revealed.” For this statement to have veracity, the Challenger must trust the policies used by the entity that issued the identity and the maintenance policy of the TPM manufacturer.

Signing and legacy keys may be used to certify both migratable and non-migratable keys. Then the usefulness of a certificate depends on the trust in the certifying key by the recipient of the certificate.

The key to be certified must be loaded before TPM\_CertifyKey is called.

The determination to use the TPM\_CERTIFY\_INFO or TPM\_CERTIFY\_INFO2 on the output is based on which PCRs and what localities the certified key is restricted to. A key to be certified that does not have locality restrictions and which uses no PCRs greater than PCR #15 will cause this command return and sign a TPM\_CERTIFY\_INFO structure, which provides compatibility with V1.1 TPMs.

When this command is run to certify all other keys (those that use PCR #16 or higher, as well as those limited by locality in any way) it will return and sign a TPM\_CERTIFY\_INFO2 structure.

TPM\_CertifyKey does not support the case where (a) the certifying key requires a usage authorization to be provided but (b) the key-to-be-certified does not. In such cases, TPM\_CertifyKey2 must be used.

If a command tag (in the parameter array) specifies only one authorisation session, then the TPM convention is that the first session listed is ignored (authDataUsage must be NEVER for this key) and the incoming session data is used for the second auth session in the list. In TPM\_CertifyKey, the first session is the certifying key and the second session is the key-to-be-certified. In TPM\_CertifyKey2, the first session is the key-to-be-certified and the second session is the certifying key.

**End of informative comment.**

## 246 Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal at TPM_ORD_CertifyKey
4	4			TPM_KEY_HANDLE	certHandle	Handle of the key to be used to certify the key.
5	4			TPM_KEY_HANDLE	keyHandle	Handle of the key to be certified.
6	20	2S	20	TPM_NONCE	antiReplay	160 bits of externally supplied data (typically a nonce provided to prevent replay -attacks)
7	4			TPM_AUTHHANDLE	certAuthHandle	The authorization session handle used for certHandle.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with certAuthHandle
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
10	20			TPM_AUTHDATA	certAuth	The authorization session digest for inputs and certHandle. HMAC key: certKey.auth.
11	4			TPM_AUTHHANDLE	keyAuthHandle	The authorization session handle used for the key to be signed.
		2H2	20	TPM_NONCE	keylastNonceEven	Even nonce previously generated by TPM
12	20	3H2	20	TPM_NONCE	keynonceOdd	Nonce generated by system associated with keyAuthHandle
13	1	4H2	1	BOOL	continueKeySession	The continue use flag for the authorization session handle
14	20			TPM_AUTHDATA	keyAuth	The authorization session digest for the inputs and key to be signed. HMAC key: key.usageAuth.



247 **Outgoing Operands and Sizes**

Param		HMAC		Type	Name	Description
#	Sz	#	Sz			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_CertifyKey
4	<>	3S	<>	TPM_CERTIFY_INFO	certifyInfo	TPM_CERTIFY_INFO or TPM_CERTIFY_INFO2 structure that provides information relative to keyhandle
5	4	4S	4	UINT32	outDataSize	The used size of the output area for outData
6	<>	5S	<>	BYTE[]	outData	The signature of certifyInfo
7	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with certAuthHandle
8	1	4H1	1	BOOL	continueAuthSession	Continue use flag for cert key session
9	20		20	TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters and parentHandle. HMAC key: certKey -> auth.
10	20	2H2	20	TPM_NONCE	keyNonceEven	Even nonce newly generated by TPM
		3H2	20	TPM_NONCE	keynonceOdd	Nonce generated by system associated with keyAuthHandle
11	1	4H2	1	BOOL	continueKey Session	Continue use flag for target key session
12	20			TPM_AUTHDATA	keyAuth	The authorization session digest for the target key. HMAC key: key.auth.

248 **Actions**

- 249 1. The TPM validates that the key pointed to by certHandle has a signature scheme of  
250 TPM\_SS\_RSASSAPKCS1v15\_SHA1
- 251 2. Verify command and key AuthData values:
- 252 a. If tag is TPM\_TAG\_RQU\_AUTH2\_COMMAND
- 253 i. The TPM verifies the AuthData in certAuthHandle provides authorization to use  
254 the key pointed to by certHandle, return TPM\_AUTHFAIL on error
- 255 ii. The TPM verifies the AuthData in keyAuthHandle provides authorization to use  
256 the key pointed to by keyHandle, return TPM\_AUTH2FAIL on error
- 257 b. else if tag is TPM\_TAG\_RQU\_AUTH1\_COMMAND
- 258 i. Verify that authDataUsage is TPM\_AUTH\_NEVER for the key referenced by  
259 certHandle, return TPM\_AUTHFAIL on error.
- 260 ii. The TPM verifies the AuthData in keyAuthHandle provides authorization to use  
261 the key pointed to by keyHandle, return TPM\_AUTHFAIL on error
- 262 c. else if tag is TPM\_TAG\_RQU\_COMMAND
- 263 i. Verify that authDataUsage is TPM\_AUTH\_NEVER for the key referenced by  
264 certHandle, return TPM\_AUTHFAIL on error.

- 265        ii. Verify that authDataUsage is TPM\_AUTH\_NEVER or TPM\_AUTH\_PRIV\_USE\_ONLY  
266            for the key referenced by keyHandle, return TPM\_AUTHFAIL on error.
- 267    3. If the key pointed to by certHandle is an identity key (certHandle -> keyUsage is  
268        TPM\_KEY\_IDENTITY)
  - 269        a. If keyHandle -> keyFlags -> migratable is TRUE return TPM\_MIGRATEFAIL
- 270    4. If keyHandle -> digestAtRelease requires the use of PCRs 16 or higher to calculate or if  
271        keyHandle -> localityAtRelease is not 0x1F
  - 272        a. Set V1 to 1.2
- 273    5. Else
  - 274        a. Set V1 to 1.1
- 275    6. If keyHandle -> pcrInfoSize is not 0
  - 276        a. If keyHandle -> keyFlags has pcrIgnoredOnRead set to FALSE
    - 277            i. Create a digestAtRelease according to the specified TPM\_STCLEAR\_DATA -> PCR  
278                registers and compare to keyHandle -> digestAtRelease and if a mismatch return  
279                TPM\_WRONGPCRVAL
    - 280            ii. If specified validate any locality requests on error TPM\_BAD\_LOCALITY
  - 281        b. If V1 is 1.1
    - 282            i. Create C1 a TPM\_CERTIFY\_INFO structure
    - 283            ii. Fill in C1 with the information from the key pointed to by keyHandle
    - 284            iii. The TPM MUST set c1 -> pcrInfoSize to 44.
    - 285            iv. The TPM MUST set c1 -> pcrInfo to a TPM\_PCR\_INFO structure properly filled out  
286                using the information from keyHandle.
    - 287            v. The TPM MUST set c1 -> digestAtCreation to 20 bytes of 0x00.
  - 288        c. Else
    - 289            i. Create C1 a TPM\_CERTIFY\_INFO2 structure
    - 290            ii. Fill in C1 with the information from the key pointed to by keyHandle
    - 291            iii. Set C1 -> pcrInfoSize to the size of an appropriate TPM\_PCR\_INFO\_SHORT  
292                structure.
    - 293            iv. Set C1 -> pcrInfo to a properly filled out TPM\_PCR\_INFO\_SHORT structure, using  
294                the information from keyHandle.
    - 295            v. Set C1 -> migrationAuthoritySize to 0
- 296    7. Else
  - 297        a. Create C1 a TPM\_CERTIFY\_INFO structure
  - 298        b. Fill in C1 with the information from the key pointed to by keyHandle
  - 299        c. The TPM MUST set c1 -> pcrInfoSize to 0
- 300    8. Create TPM\_DIGEST H1 which is the SHA-1 hash of keyHandle -> pubKey -> key. Note  
301        that <key> is the actual public modulus, and does not include any structure formatting.

- 302 9. Set C1 -> pubKeyDigest to H1
- 303 10. The TPM copies the antiReplay parameter to c1 -> data.
- 304 11. The TPM sets certifyInfo to C1.
- 305 12. The TPM creates m1, a message digest formed by taking the SHA1 of c1.
- 306 a. The TPM then computes a signature using certHandle -> sigScheme. The resulting
- 307 signed blob is returned in outData.

## 13.9 TPM\_CertifyKey2

### Start of informative comment:

This command is based on TPM\_CertifyKey, but includes the ability to certify a Certifiable Migration Key (CMK), which requires extra input parameters.

TPM\_CertifyKey2 always produces a TPM\_CERTIFY\_INFO2 structure.

TPM\_CertifyKey2 does not support the case where (a) the key-to-be-certified requires a usage authorization to be provided but (b) the certifying key does not.

If a command tag (in the parameter array) specifies only one authorisation session, then the TPM convention is that the first session listed is ignored (authDataUsage must be NEVER for this key) and the incoming session data is used for the second auth session in the list. In TPM\_CertifyKey2, the first session is the key to be certified and the second session is the certifying key.

### End of informative comment.

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_CertifyKey2
4	4			TPM_KEY_HANDLE	keyHandle	Handle of the key to be certified.
5	4			TPM_KEY_HANDLE	certHandle	Handle of the key to be used to certify the key.
6	20	2S	20	TPM_DIGEST	migrationPubDigest	The digest of a TPM_MSA_COMPOSITE structure, containing at least one public key of a Migration Authority
7	20	3S	20	TPM_NONCE	antiReplay	160 bits of externally supplied data (typically a nonce provided to prevent replay -attacks)
8	4			TPM_AUTHHANDLE	keyAuthHandle	The authorization session handle used for the key to be signed.
		2H1	20	TPM_NONCE	keylastNonceEven	Even nonce previously generated by TPM
9	20	3H1	20	TPM_NONCE	keynonceOdd	Nonce generated by system associated with keyAuthHandle
10	1	4H1	1	BOOL	continueKeySession	The continue use flag for the authorization session handle
11	20			TPM_AUTHDATA	keyAuth	The authorization session digest for the inputs and key to be signed. HMAC key: key.usageAuth.
12	4			TPM_AUTHHANDLE	certAuthHandle	The authorization session handle used for certHandle.
		2H2	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
13	20	3H2	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with certAuthHandle
14	1	4H2	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
15	20			TPM_AUTHDATA	certAuth	Authorization HMAC key: certKey.auth.

322 **Outgoing Operands and Sizes**

Param		HMAC		Type	Name	Description
#	Sz	#	Sz			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_CertifyKey2
4	◇	3S	◇	TPM_CERTIFY_INFO2	certifyInfo	TPM_CERTIFY_INFO2 relative to keyHandle
5	4	4S	4	UINT32	outDataSize	The used size of the output area for outData
6	◇	5S	◇	BYTE[]	outData	The signed public key.
7	20	2H1	20	TPM_NONCE	keyNonceEven	Even nonce newly generated by TPM
		3H1	20	TPM_NONCE	keyNonceOdd	Nonce generated by system associated with certAuthHandle
8	1	4H1	1	BOOL	keyContinueAuthSession	Continue use flag for cert key session
9	20		20	TPM_AUTHDATA	keyResAuth	Authorization HMAC key: keyHandle -> auth.
10	20	2H2	20	TPM_NONCE	certNonceEven	Even nonce newly generated by TPM
		3H2	20	TPM_NONCE	AuthLastNonceOdd	Nonce generated by system associated with certAuthHandle
11	1	4H2	1	BOOL	CertContinueAuthSession	Continue use flag for cert key session
12	20		20	TPM_AUTHDATA	certResAuth	Authorization HMAC key: certHandle -> auth.

323 **Actions**

- 324 1. The TPM validates that the key pointed to by certHandle has a signature scheme of  
325 TPM\_SS\_RSASSAPKCS1v15\_SHA1
- 326 2. Verify command and key AuthData values:
  - 327 a. If tag is TPM\_TAG\_RQU\_AUTH2\_COMMAND
    - 328 i. The TPM verifies the AuthData in keyAuthHandle provides authorization to use  
329 the key pointed to by keyHandle, return TPM\_AUTHFAIL on error
    - 330 ii. The TPM verifies the AuthData in certAuthHandle provides authorization to use  
331 the key pointed to by certHandle, return TPM\_AUTH2FAIL on error
  - 332 b. else if tag is TPM\_TAG\_RQU\_AUTH1\_COMMAND
    - 333 i. Verify that authDataUsage is TPM\_AUTH\_NEVER or TPM\_AUTH\_PRIV\_USE\_ONLY  
334 for the key referenced by keyHandle, return TPM\_AUTHFAIL on error
    - 335 ii. The TPM verifies the AuthData in certAuthHandle provides authorization to use  
336 the key pointed to by certHandle, return TPM\_AUTH2FAIL on error
  - 337 c. else if tag is TPM\_TAG\_RQU\_COMMAND
    - 338 i. Verify that authDataUsage is TPM\_AUTH\_NEVER or TPM\_AUTH\_PRIV\_USE\_ONLY  
339 for the key referenced by keyHandle, return TPM\_AUTHFAIL on error
    - 340 ii. Verify that authDataUsage is TPM\_AUTH\_NEVER for the key referenced by  
341 certHandle, return TPM\_AUTHFAIL on error.

342 3. If the key pointed to by certHandle is an identity key (certHandle -> keyUsage is  
343 TPM\_KEY\_IDENTITY)  
344 a. If keyHandle -> keyFlags -> migratable is TRUE and [keyHandle -> keyFlags->  
345 migrateAuthority is FALSE or (keyHandle -> payload != TPM\_PT\_MIGRATE\_RESTRICTED  
346 and keyHandle -> payload != TPM\_PT\_MIGRATE\_EXTERNAL)] return  
347 TPM\_MIGRATEFAIL  
348 4. The TPM SHALL create a c1 a TPM\_CERTIFY\_INFO2 structure from the key pointed to  
349 by keyHandle  
350 5. Create TPM\_DIGEST H1 which is the SHA-1 hash of keyHandle -> pubKey -> key. Note  
351 that <key> is the actual public modulus, and does not include any structure formatting.  
352 6. Set C1 -> pubKeyDigest to H1  
353 7. Copy the antiReplay parameter to c1 -> data  
354 8. Copy other keyHandle parameters into C1  
355 9. If keyHandle -> payload == TPM\_PT\_MIGRATE\_RESTRICTED or  
356 TPM\_PT\_MIGRATE\_EXTERNAL  
357 a. create thisPubKey, a TPM\_PUBKEY structure containing the public key, algorithm  
358 and parameters corresponding to keyHandle  
359 b. Verify that the migration authorization is valid for this key  
360 i. Create M2 a TPM\_CMK\_MIGAUTH structure  
361 ii. Set M2 -> msaDigest to migrationPubDigest  
362 iii. Set M2 -> pubkeyDigest to SHA1[thisPubKey]  
363 iv. Verify that [keyHandle -> migrationAuth] == HMAC(M2) signed by using tpmProof  
364 as the secret and return error TPM\_MA\_SOURCE on mismatch  
365 c. Set C1 -> migrationAuthority = SHA-1(migrationPubDigest || keyHandle -> payload)  
366 d. if keyHandle -> payload == TPM\_PT\_MIGRATE\_RESTRICTED  
367 i. Set C1 -> payloadType = TPM\_PT\_MIGRATE\_RESTRICTED  
368 e. if keyHandle -> payload == TPM\_PT\_MIGRATE\_EXTERNAL  
369 i. Set C1 -> payloadType = TPM\_PT\_MIGRATE\_EXTERNAL  
370 10.Else  
371 a. set C1 -> migrationAuthority = NULL  
372 b. set C1 -> migrationAuthoritySize =0  
373 c. Set C1 -> payloadType = TPM\_PT\_ASYM  
374 11.If keyHandle -> pcrInfoSize is not 0  
375 a. The TPM MUST set c1 -> pcrInfoSize to match the pcrInfoSize from the keyHandle  
376 key.  
377 b. The TPM MUST set c1 -> pcrInfo to match the pcrInfo from the keyHandle key  
378 c. If keyHandle -> keyFlags has pcrIgnoredOnRead set to FALSE

379 i. Create a digestAtRelease according to the specified TPM\_STCLEAR\_DATA -> PCR  
380 registers and compare to keyHandle -> digestAtRelease and if a mismatch return  
381 TPM\_WRONGPCRVAL

382 ii. If specified validate any locality requests on error TPM\_BAD\_LOCALITY

383 12.Else

384 a. The TPM MUST set c1 -> pcrInfoSize to 0

385 13.The TPM creates m1, a message digest formed by taking the SHA1 of c1

386 a. The TPM then computes a signature using certHandle -> sigScheme. The resulting  
387 signed blob is returned in outData

## 14. Endorsement Key Handling

### **Start of informative comment:**

There are two create EK commands. The first matches the 1.1 functionality. The second provides the mechanism to enable revokeEK.

The TPM and platform manufacturer decide on the inclusion or exclusion of the ability to execute revokeEK.

The restriction to have the TPM generate the EK does not remove the manufacturing option to “squirt” the EK. During manufacturing, the TPM does not enforce all protections or requirements; hence, the restriction on only TPM generation of the EK is also not in force.

### **End of informative comment.**

1. A TPM SHALL NOT install an EK unless generated on the TPM by execution of TPM\_CreateEndorsementKeyPair or TPM\_CreateRevocableEK



**14.1 TPM\_CreateEndorsementKeyPair****Start of informative comment:**

This command creates the TPM endorsement key. It returns a failure code if an endorsement key already exists.

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateEndorsementKeyPair
4	20	2S	20	TPM_NONCE	antiReplay	Arbitrary data
5	<>	3S	<>	TPM_KEY_PARMS	keyInfo	Information about key to be created, this includes all algorithm parameters

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateEndorsementKeyPair
4	<>	3S	<>	TPM_PUBKEY	pubEndorsementKey	The public endorsement key
5	20	4S	20	TPM_DIGEST	checksum	Hash of pubEndorsementKey and antiReplay

**Actions**

1. If an EK already exists, return TPM\_DISABLED\_CMD
2. Validate the keyInfo parameters for the key description
  - a. If the algorithm type is RSA the key length MUST be a minimum of 2048. For interoperability the key length SHOULD be 2048
  - b. If the algorithm type is other than RSA the strength provided by the key MUST be comparable to RSA 2048
  - c. The other parameters of keyInfo (signatureScheme etc.) are ignored.
3. Create a key pair called the “endorsement key pair” using a TPM-protected capability. The type and size of key are that indicated by keyInfo
4. Create checksum by performing SHA1 on the concatenation of (PUBEK || antiReplay)
5. Store the PRIVEK
6. Create TPM\_PERMANENT\_DATA -> tpmDAASeed from the TPM RNG

- 420 7. Set TPM\_PERMANENT\_FLAGS -> CEKPU sed to TRUE
- 421 8. Set TPM\_PERMANENT\_FLAGS -> enableRevokeEK to FALSE

## 14.2 TPM\_CreateRevocableEK

**Start of informative comment:**

This command creates the TPM endorsement key. It returns a failure code if an endorsement key already exists. The TPM vendor may have a separate mechanism to create the EK and “squirt” the value into the TPM.

The input parameters specify whether the EK is capable of being reset, whether the AuthData value to reset the EK will be generated by the TPM, and the new AuthData value itself if it is not to be generated by the TPM. The output parameter is the new AuthData value that must be used when resetting the EK (if it is capable of being reset).

The command TPM\_RevokeTrust must be used to reset an EK (if it is capable of being reset).

Owner authorisation is unsuitable for authorizing resetting of an EK: someone with Physical Presence can remove a genuine Owner, install a new Owner, and revoke the EK. The genuine Owner can reinstall, but the platform will have lost its original attestation and may not be trusted by challengers. Therefore if a password is to be used to revoke an EK, it must be a separate password, given to the genuine Owner.

In v1.2 an OEM has extra choices when creating EKs.

a) An OEM could manufacture all of its TPMs with enableRevokeEK==TRUE.

If the OEM has tracked the EKreset passwords for these TPMs, the OEM can give the passwords to customers. The customers can use the passwords as supplied, change the passwords, or clear the EKs and create new EKs with new passwords.

If EKreset passwords are random values, the OEM can discard those values and not give them to customers. There is then a low probability (statistically zero) chance of a local DOS attack to reset the EK by guessing the password. The chance of a remote DOS attack is zero because Physical Presence must also be asserted to use TPM\_RevokeTrust.

b) An OEM could manufacture some of its TPMs with enableRevokeEK==FALSE. Then the EK can never be revoked, and the chance of even a local DOS attack on the EK is eliminated.

**End of informative comment.**

This is an optional command

## 452 Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateRevocableEK
4	20	2S	20	TPM_NONCE	antiReplay	Arbitrary data
5	◇	3S	◇	TPM_KEY_PARMS	keyInfo	Information about key to be created, this includes all algorithm parameters
6	1	4S	1	BOOL	generateReset	If TRUE use TPM RNG to generate EKreset. If FALSE use the passed value inputEKreset
7	20	5S	20	TPM_NONCE	inputEKreset	The authorization value to be used with TPM_RevokeTrust if generateReset==FALSE, else the parameter is present but ignored

## 453 Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateRevocableEK
4	◇	3S	◇	TPM_PUBKEY	pubEndorsementKey	The public endorsement key
5	20	4S	20	TPM_DIGEST	checksum	Hash of pubEndorsementKey and antiReplay
6	20	5S	20	TPM_NONCE	outputEKreset	The AuthData value to use TPM_RevokeTrust

## 454 Actions

- 455 1. If an EK already exists, return TPM\_DISABLED\_CMD
- 456 2. Perform the actions of TPM\_CreateEndorsementKeyPair, if any errors return with error
- 457 3. Set TPM\_PERMANENT\_FLAGS -> enableRevokeEK to TRUE
  - 458 a. If generateReset is TRUE then
    - 459 i. Set TPM\_PERMANENT\_DATA -> EKreset to the next value from the TPM RNG
  - 460 b. Else
    - 461 i. Set TPM\_PERMANENT\_DATA -> EKreset to inputEKreset
- 462 4. Return PUBEK, checksum and Ekreset
- 463 5. Create TPM\_PERMANENT\_DATA -> tpmDAASeed from the TPM RNG
- 464 6. The outputEKreset AuthData is sent in the clear. There is no uniqueness on the TPM
- 465 available to actually perform encryption or use an encrypted channel. The assumption is
- 466 that this operation is occurring in a controlled environment and sending the value in the
- 467 clear is acceptable.

### 14.3 TPM\_RevokeTrust

**Start of informative comment:**

This command clears the EK and sets the TPM back to a pure default state. The generation of the AuthData value occurs during the generation of the EK. It is the responsibility of the EK generator to properly protect and disseminate the RevokeTrust AuthData.

**End of informative comment.**

This is an optional command

#### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_RevokeTrust
4	20	2S	20	TPM_DIGEST	EKReset	The value that will be matched to EK Reset

#### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_RevokeTrust

#### Actions

1. The TPM MUST validate that TPM\_PERMANENT\_FLAGS -> enableRevokeEK is TRUE, return TPM\_PERMANENTEK on error
2. The TPM MUST validate that the EKReset matches TPM\_PERMANENT\_DATA -> EKReset return TPM\_AUTHFAIL on error.
3. Ensure that physical presence is being asserted
4. Perform the actions of TPM\_OwnerClear (excepting the command authentication)
  - a. NV items with the pubInfo -> nvIndex D value set MUST be deleted. This changes the TPM\_OwnerClear handling of the same NV areas
  - b. Set TPM\_PERMANENT\_FLAGS -> nvLocked to FALSE
5. Invalidate TPM\_PERMANENT\_DATA -> tpmDAASeed
6. Invalidate the EK and any internal state associated with the EK

## 14.4 TPM\_ReadPubek

### Start of informative comment:

Return the endorsement key public portion. This value should have controls placed upon access, as it is a privacy sensitive value.

The readPubek flag is set to FALSE by TPM\_TakeOwnership and set to TRUE by TPM\_OwnerClear, thus mirroring if a TPM Owner is present.

### End of informative comment.

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReadPubek
4	20	2S	20	TPM_NONCE	antiReplay	Arbitrary data

### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReadPubek
4	<>	3S	<>	TPM_PUBKEY	pubEndorsementKey	The public endorsement key
5	20	4S	20	TPM_DIGEST	checksum	Hash of pubEndorsementKey and antiReplay

### Description

This command returns the PUBEK.

### Actions

The TPM\_ReadPubek command SHALL

1. If TPM\_PERMANENT\_FLAGS -> readPubek is FALSE return TPM\_DISABLED\_CMD
2. If no EK is present the TPM MUST return TPM\_NO\_ENDORSEMENT
3. Create checksum by performing SHA1 on the concatenation of (pubEndorsementKey || antiReplay).
4. Export the PUBEK and checksum.

**14.5 TPM\_OwnerReadInternalPub****Start of informative comment:**

A TPM Owner authorized command that returns the public portion of the EK or SRK.

The keyHandle parameter is included in the incoming session authorization to prevent alteration of the value, causing a different key to be read. Unlike most key handles, which can be mapped by higher layer software, this key handle has only two fixed values.

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OwnerReadInternalPub
4	4	2S	4	TPM_KEY_HANDLE	keyHandle	Handle for either PUBEK or SRK
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
8	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner authentication. HMAC key: ownerAuth.

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OwnerReadInternalPub
4	◇	3S	◇	TPM_PUBKEY	publicPortion	The public portion of the requested key
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

**Actions**

1. Validate the parameters and TPM Owner AuthData for this command
2. If keyHandle is TPM\_KH\_EK
  - a. Set publicPortion to PUBEK

- 520 3. Else If keyHandle is TPM\_KH\_SRK
- 521     a. Set publicPortion to the TPM\_PUBKEY of the SRK
- 522 4. Else return TPM\_BAD\_PARAMETER
- 523 5. Export the public key of the referenced key



## 15. Identity Creation and Activation

### 15.1 TPM\_MakeIdentity

**Start of informative comment:**

Generate a new Attestation Identity Key (AIK)

**End of informative comment.**

#### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of input bytes incl. paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_MakeIdentity.
4	20	2S	20	TPM_ENCAUTH	identityAuth	Encrypted usage AuthData for the new identity
5	20	3S	20	TPM_CHOSENID_HASH	labelPrivCADigest	The digest of the identity label and privacy CA chosen for the AIK
6	◇	4S	◇	TPM_KEY	idKeyParams	Structure containing all parameters of new identity key. pubKey.keyLength & idKeyParams.encData are both 0MAY be TPM_KEY12
7	4			TPM_AUTHHANDLE	srkAuthHandle	The authorization session handle used for SRK authorization.
		2H1	20	TPM_NONCE	srkLastNonceEven	Even nonce previously generated by TPM
8	20	3H1	20	TPM_NONCE	srknonceOdd	Nonce generated by system associated with srkAuthHandle
9	1	4H1	1	BOOL	continueSrSession	Ignored
10	20			TPM_AUTHDATA	srkAuth	The authorization session digest for the inputs and the SRK. HMAC key: srk.usageAuth.
11	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication. Session type MUST be OSAP.
		2H2	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
12	20	3H2	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
13	1	4H2	1	BOOL	continueAuthSession	Ignored
14	20		20	TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner. HMAC key: ownerAuth.

## 530 Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_MakeIdentity.
4	◇	3S	◇	TPM_KEY	idKey	The newly created identity key . MAY be TPM_KEY12
5	4	4S	4	UINT32	identityBindingSize	The used size of the output area for identityBinding
6	◇	5S	◇	BYTE[]	identityBinding	Signature of TPM_IDENTITY_CONTENTS using idKey.private.
7	20	2H2	20	TPM_NONCE	srkNonceEven	Even nonce newly generated by TPM.
		3H2	20	TPM_NONCE	srknonceOdd	Nonce generated by system associated with srkAuthHandle
8	1	4H2	1	BOOL	continueSrkJession	Continue use flag. Fixed value of FALSE
9	20			TPM_AUTHDATA	srkAuth	The authorization session digest used for the outputs and srkAuth session. HMAC key: srk.usageAuth.
10	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
11	1	4H1	1	BOOL	continueAuthSession	Continue use flag. Fixed value of FALSE
12	20		20	TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

## 531 Description

532 The public key of the new TPM identity SHALL be identityPubKey. The private key of the  
533 new TPM identity SHALL be tpm\_signature\_key.

## 534 Properties of the new identity

Type	Name	Description
TPM_PUBKEY	identityPubKey	This SHALL be the public key of a previously unused asymmetric key pair.
TPM_STORE_ASYMKEY	tpm_signature_key	This SHALL be the private key that forms a pair with identityPubKey and SHALL be extant only in a TPM-shielded location.

535

536 This capability also generates a TPM\_KEY containing the tpm\_signature\_key.

537 If identityPubKey is stored on a platform it SHALL exist only in storage to which access is  
538 controlled and is available to authorized entities.

## 539 Actions

540 A Trusted Platform Module that receives a valid TPM\_MakeIdentity command SHALL do the  
541 following:

- 542 1. Validate the idKeyParams parameters for the key description
  - 543 a. If the algorithm type is RSA the key length MUST be a minimum of 2048. For  
544 interoperability the key length SHOULD be 2048

- 545       b. If the algorithm type is other than RSA the strength provided by the key MUST be
- 546       comparable to RSA 2048
- 547       c. If the TPM is not designed to create a key of the requested type, return the error code
- 548       TPM\_BAD\_KEY\_PROPERTY
- 549       d. If TPM\_PERMANENT\_FLAGS -> FIPS is TRUE then
- 550           i. If authDataUsage specifies TPM\_AUTH\_NEVER return TPM\_NOTFIPS
- 551   2. Use authHandle to verify that the Owner authorized all TPM\_MakeIdentity input
- 552       parameters.
- 553   3. Use srkAuthHandle to verify that the SRK owner authorized all TPM\_MakeIdentity input
- 554       parameters.
- 555   4. Verify that idKeyParams -> keyUsage is TPM\_KEY\_IDENTITY. If it is not, return
- 556       TPM\_INVALID\_KEYUSAGE
- 557   5. Verify that idKeyParams -> keyFlags -> migratable is FALSE. If it is not, return
- 558       TPM\_INVALID\_KEYUSAGE
- 559   6. If authHandle indicates XOR encryption for the AuthData secrets
- 560       a. Create X1 the SHA-1 of the concatenation of (ownerAuth -> sharedSecret ||
- 561       authLastNonceEven)
- 562       b. Create a1 by XOR X1 and identityAuth
- 563   7. Else
- 564       a. Create a1 by decrypting identityAuth using the algorithm indicated in the OSAP
- 565       session
- 566       b. Key is from ownerAuth -> sharedSecret
- 567       c. IV is SHA-1 of (authLastNonceEven || nonceOdd)
- 568   8. Set continueAuthSession and continueSRKSession to FALSE.
- 569   9. Determine the structure version
- 570       a. If idKeyParams -> tag is TPM\_TAG\_KEY12
- 571           i. Set V1 to 2
- 572           ii. Create idKey a TPM\_KEY12 structure using idKeyParams as the default values for
- 573               the structure
- 574       b. If idKeyParams -> ver is 1.1
- 575           i. Set V1 to 1
- 576           ii. Create idKey a TPM\_KEY structure using idKeyParams as the default values for
- 577               the structure
- 578   10. Set the digestAtCreation values for pcrInfo
- 579       a. For TPM\_PCR\_INFO\_LONG include the locality of the current command
- 580   11. Create an asymmetric key pair (identityPubKey and tpm\_signature\_key) using a TPM-
- 581       protected capability, in accordance with the algorithm specified in idKeyParams

- 582 12.Ensure that the AuthData information in A1 is properly stored in the idKey as  
583 usageAuth.
- 584 13.Attach identityPubKey and tpm\_signature\_key to idKey
- 585 14.Set idKey -> migrationAuth to TPM\_PERMANENT\_DATA-> tpmProof
- 586 15.Ensure that all TPM\_PAYLOAD\_TYPE structures identify this key as TPM\_PT\_ASYM
- 587 16.Encrypt the private portion of idKey using the SRK as the parent key
- 588 17.Create a TPM\_IDENTITY\_CONTENTS structure named idContents using  
589 labelPrivCADigest and the information from idKey
- 590 18.Sign idContents using tpm\_signature\_key and TPM\_SS\_RSASSAPKCS1v15\_SHA1. Store  
591 the result in identityBinding.

**15.2 TPM\_ActivateIdentity****Start of informative comment:**

The purpose of TPM\_ActivateIdentity is twofold. The first purpose is to obtain assurance that the credential in the TPM\_SYM\_CA\_ATTESTATION is for this TPM. The second purpose is to obtain the session key used to encrypt the TPM\_IDENTITY\_CREDENTIAL.

This is an extension to the 1.1 functionality of TPM\_ActivateIdentity. The blob sent to from the CA can be in the 1.1 format or the 1.2 format. The TPM determines the type from the size or version information in the blob.

TPM\_ActivateIdentity checks that the symmetric session key corresponds to a TPM-identity before releasing that session key.

Only the Owner of the TPM has the privilege of activating a TPM identity. The Owner is required to authorize the TPM\_ActivateIdentity command. The owner may authorize the command using either the TPM\_OIAP or TPM\_OSAP authorization protocols.

The creator of the ActivateIdentity package can specify if any PCR values are to be checked before releasing the session key.

**End of informative comment.****Incoming Parameters and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of input bytes incl. paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ActivateIdentity
4	4			TPM_KEY_HANDLE	idKey Handle	Identity key to be activated
5	4	2S	4	UINT32	blobSize	Size of encrypted blob from CA
6	◇	3S	◇	BYTE []	blob	The encrypted ASYM_CA_CONTENTS or TPM_EK_BLOB
7	4			TPM_AUTHHANDLE	idKeyAuthHandle	The authorization session handle used for ID key authorization.
		2H1	20	TPM_NONCE	idKeyLastNonceEven	Even nonce previously generated by TPM
8	20	3H1	20	TPM_NONCE	idKeynonceOdd	Nonce generated by system associated with idKeyAuthHandle
9	1	4H1	1	BOOL	continueIdKeySession	Continue usage flag for idKeyAuthHandle.
10	20			TPM_AUTHDATA	idKeyAuth	The authorization session digest for the inputs and ID key. HMAC key: idKey.usageAuth.
11	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H2	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
12	20	3H2	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
13	1	4H2	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
14	20		20	TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner. HMAC key: ownerAuth.

## 609 Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal:TPM_ORD_ActivateIdentity
4	◇	3S	◇	TPM_SYMMETRIC_KEY	symmetricKey	The decrypted symmetric key.
5	20	2H1	20	TPM_NONCE	idKeyNonceEven	Even nonce newly generated by TPM.
		3H1	20	TPM_NONCE	idKeynonceOdd	Nonce generated by system associated with idKeyAuthHandle
6	1	4H1	1	BOOL	continueIdKeySession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	idKeyAuth	The authorization session digest used for the returned parameters and idKeyAuth session. HMAC key: idKey.usageAuth.
8	20	2H2	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H2	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H2	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
10	20		20	TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

## 610 Description

- 611 1. The command TPM\_ActivateIdentity activates a TPM identity created using the command  
612 TPM\_MakeIdentity.
- 613 2. The command assumes the availability of the private key associated with the identity.  
614 The command will verify the association between the keys during the process.
- 615 3. The command will decrypt the input blob and extract the session key and verify the  
616 connection between the public and private keys. The input blob can be in 1.1 or 1.2  
617 format.

## 618 Actions

- 619 A Trusted Platform Module that receives a valid TPM\_ActivateIdentity command SHALL do  
620 the following:
- 621 1. Using the authHandle field, validate the owner's AuthData to execute the command and  
622 all of the incoming parameters.
  - 623 2. Using the idKeyAuthHandle, validate the AuthData to execute command and all of the  
624 incoming parameters
  - 625 3. Validate that the idKey is the public key of a valid TPM identity by checking that  
626 idKeyHandle -> keyUsage is TPM\_KEY\_IDENTITY. Return TPM\_BAD\_PARAMETER on  
627 mismatch
  - 628 4. Create H1 the digest of a TPM\_PUBKEY derived from idKey
  - 629 5. Decrypt blob creating B1 using PRIVEK as the decryption key

- 630 6. Determine the type and version of B1
- 631 a. If B1 -> tag is TPM\_TAG\_EK\_BLOB then
- 632 i. B1 is a TPM\_EK\_BLOB
- 633 b. Else
- 634 i. B1 is a TPM\_ASYM\_CA\_CONTENTS. As there is no tag for this structure it is
- 635 possible for the TPM to make a mistake here but other sections of the structure
- 636 undergo validation
- 637 7. If B1 is a version 1.1 TPM\_ASYM\_CA\_CONTENTS then
- 638 a. Compare H1 to B1 -> idDigest on mismatch return TPM\_BAD\_PARAMETER
- 639 b. Set K1 to B1 -> sessionKey
- 640 8. If B1 is a TPM\_EK\_BLOB then
- 641 a. Validate that B1 -> ekType is TPM\_EK\_TYPE\_ACTIVATE, return TPM\_BAD\_TYPE if
- 642 not.
- 643 b. Assign A1 as a TPM\_EK\_BLOB\_ACTIVATE structure from B1 -> blob
- 644 c. Compare H1 to A1 -> idDigest on mismatch return TPM\_BAD\_PARAMETER
- 645 d. If A1 -> pcrSelection is not NULL
- 646 i. Compute a composite hash C1 using the PCR selection A1 -> pcrSelection
- 647 ii. Compare C1 to A1 -> pcrInfo->digestAtRelease and return TPM\_WRONGPCRVAL
- 648 on a mismatch
- 649 iii. If A1 -> pcrInfo specifies a locality ensure that the appropriate locality has been
- 650 asserted, return TPM\_BAD\_LOCALITY on error
- 651 e. Set K1 to A1 -> symmetricKey
- 652 9. Return K1

## 16. Integrity Collection and Reporting

**Start of informative comment:**

This section deals with what commands have direct access to the PCR

**End of informative comment.**

1. The TPM SHALL only allow the following commands to alter the value of TPM\_STCLEAR\_DATA -> PCR
  - a. TPM\_Extend
  - b. TPM\_SHA1CompleteExtend
  - c. TPM\_Startup
  - d. TPM\_PCR\_Reset



**16.1 TPM\_Extend****Start of informative comment:**

This adds a new measurement to a PCR

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_Extend.
4	4	2S	4	TPM_PCRINDEX	pcrNum	The PCR to be updated.
5	20	3S	20	TPM_DIGEST	inDigest	The 160 bit value representing the event to be recorded.

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_Extend.
4	20	3S	20	TPM_PCRVALUE	outDigest	The PCR value after execution of the command.

**Descriptions**

Add a measurement value to a PCR

**Actions**

1. Map L1 to TPM\_STANY\_FLAGS -> localityModifier
2. Map P1 to TPM\_PERMANENT\_DATA -> pcrAttrib [pcrNum]. pcrExtendLocal
3. If, for the value of L1, the corresponding bit is not set in the bit map P1, return TPM\_BAD\_LOCALITY
4. Create c1 by concatenating (TPM\_STCLEAR\_DATA -> PCR[pcrNum] || inDigest). This takes the current PCR value and concatenates the inDigest parameter.
5. Create h1 by performing a SHA1 digest of c1.
6. Store h1 to TPM\_STCLEAR\_DATA -> PCR[pcrNum]
7. If TPM\_PERMANENT\_FLAGS -> disable is TRUE or TPM\_STCLEAR\_FLAGS -> deactivated is TRUE
  - a. Set outDigest to 20 bytes of 0x00

- 683 8. Else
- 684 a. Set outDigest to h1

**16.2 TPM\_PCRRead****Start of informative comment:**

The TPM\_PCRRead operation provides non-cryptographic reporting of the contents of a named PCR.

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PCRRead
4	4	2S	4	TPM_PCRINDEX	pcrIndex	Index of the PCR to be read

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PCRRead
4	20	3S	20	TPM_PCRVALUE	outDigest	The current contents of the named PCR

**Description**

The TPM\_PCRRead operation returns the current contents of the named register to the caller.

**Actions**

1. Set outDigest to TPM\_STCLEAR\_DATA -> PCR[pcrIndex]
2. Return TPM\_SUCCESS

## 16.3 TPM\_Quote

### Start of informative comment:

The TPM\_Quote operation provides cryptographic reporting of PCR values. A loaded key is required for operation. TPM\_Quote uses a key to sign a statement that names the current value of a chosen PCR and externally supplied data (which may be a nonce supplied by a Challenger).

The term "ExternalData" is used because an important use of TPM\_Quote is to provide a digital signature on arbitrary data, where the signature includes the PCR values of the platform at time of signing. Hence the "ExternalData" is not just for anti-replay purposes, although it is (of course) used for that purpose in an integrity challenge.

### End of informative comment.

## Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_Quote.
4	4			TPM_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key that can sign the PCR values.
5	20	2S	20	TPM_NONCE	externalData	160 bits of externally supplied data (typically a nonce provided by a server to prevent replay -attacks)
6	<>	3S	<>	TPM_PCR_SELECTION	targetPCR	The indices of the PCRs that are to be reported.
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
10	20			TPM_AUTHDATA	privAuth	The authorization session digest for inputs and keyHandle. HMAC key: key -> usageAuth.

711 **Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_Quote.
4	◇	3S	◇	TPM_PCR_COMPOSITE	pcrData	A structure containing the same indices as targetPCR, plus the corresponding current PCR values.
5	4	4S	4	UINT32	sigSize	The used size of the output area for the signature
6	◇	5S	◇	BYTE[]	sig	The signed data blob.
7	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
9	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: Key -> usageAuth.

712 **Actions**

- 713 1. The TPM MUST validate the AuthData to use the key pointed to by keyHandle.
- 714 2. The keyHandle -> sigScheme MUST use SHA-1, return TPM\_INAPPROPRIATE\_SIG if it  
715 does not
- 716 3. Validate that keyHandle -> keyUsage is TPM\_KEY\_SIGNING, TPM\_KEY\_IDENTITY, or  
717 TPM\_KEY\_LEGACY, if not return TPM\_INVALID\_KEYUSAGE
- 718 4. Validate targetPCR
  - 719 a. targetPCR is a valid TPM\_PCR\_SELECTION structure
  - 720 b. On errors return TPM\_INVALID\_PCR\_INFO
- 721 5. Create H1 a SHA-1 hash of a TPM\_PCR\_COMPOSITE using the TPM\_STCLEAR\_DATA ->  
722 PCR indicated by targetPCR -> pcrSelect
- 723 6. Create Q1 a TPM\_QUOTE\_INFO structure
  - 724 a. Set Q1 -> version to 1.1.0.0
  - 725 b. Set Q1 -> fixed to "QUOT"
  - 726 c. Set Q1 -> digestValue to H1
  - 727 d. Set Q1 -> externalData to externalData
- 728 7. Sign SHA-1 hash of Q1 using keyHandle as the signature key
- 729 8. Return the signature in sig

## 16.4 TPM\_PCR\_Reset

### Start of informative comment:

For PCR with the pcrReset attribute set to TRUE, this command resets the PCR back to the default value, this mimics the actions of TPM\_Init. The PCR may have restrictions as to which locality can perform the reset operation.

Sending a null pcrSelection results in an error is due to the requirement that the command actually do something. If pcrSelection is null there are no PCR to reset and the command would then do nothing.

For PCR that are resettable, the presence of a Trusted Operating System (TOS) can change the behavior of TPM\_PCR\_Reset. The following pseudo code shows how the behavior changes

At TPM\_Startup

If TPM\_PCR\_ATTRIBUTES->pcrReset is FALSE

Set PCR to 0x00...00

Else

Set PCR to 0xFF...FF

At TPM\_PCR\_Reset

If TPM\_PCR\_ATTRIBUTES->pcrReset is TRUE

If TOSPresent

Set PCR to 0x00...00

Else

Set PCR to 0xFF...FF

Else

Return error

The above pseudocode is for example only, for the details of a specific platform, the reader must review the platform specific specification. The purpose of the above pseudocode is to show that both pcrReset and the TOSPresent bit control the value in use to when the PCR resets.

### End of informative comment.

## Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PCR_Reset
4	<	2S	<	TPM_PCR_SELECTION	pcrSelection	The PCR's to reset

**760 Outgoing Parameters and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PCR_Reset

**761 Descriptions**

762 This command resets PCR values back to the default value. The command MUST validate  
 763 that all PCR registers that are selected are available to be reset before resetting any PCR.  
 764 This command MUST either reset all selected PCR registers or none of the PCR registers.

**765 Actions**

- 766 1. Validate that pcrSelection is valid
  - 767 a. is a valid TPM\_PCR\_SELECTION structure
  - 768 b. pcrSelection -> pcrSelect is non-zero
  - 769 c. On errors return TPM\_INVALID\_PCR\_INFO
- 770 2. Map L1 to TPM\_STANY\_FLAGS -> localityModifier
- 771 3. For each PCR selected perform the following
  - 772 a. If TPM\_PERMANENT\_DATA -> pcrAttrib[pcrIndex].pcrReset is FALSE, return  
 773 TPM\_NOTRESETABLE
  - 774 b. If, for the value L1, the corresponding bit is clear in the bit map  
 775 TPM\_PERMANENT\_DATA -> pcrAttrib[pcrIndex].pcrResetLocal, return TPM\_NOTLOCAL
- 776 4. For each PCR selected perform the following
  - 777 a. The PCR MAY only reset to 0x00...00 or 0xFF...FF
  - 778 b. The logic to determine which value to use MUST be described by a platform specific  
 779 specification

## 16.5 TPM\_Quote2

### Start of informative comment:

The TPM\_Quote2 operation provides cryptographic reporting of PCR values. A loaded key is required for operation. TPM\_Quote2 uses a key to sign a statement that names the current value of a chosen PCR and externally supplied data (which may be a nonce supplied by a Challenger).

The term "externalData" is used because an important use of TPM\_Quote2 is to provide a digital signature on arbitrary data, where the signature includes the PCR values of the platform at time of signing. Hence the "externalData" is not just for anti-replay purposes, although it is (of course) used for that purpose in an integrity challenge.

TPM\_Quote2 differs from TPM\_Quote in that TPM\_Quote2 uses TPM\_PCR\_INFO\_SHORT to hold information relative to the PCR registers. TPM\_PCR\_INFO\_SHORT includes locality information to provide the requestor a more complete view of the current platform configuration.

### End of informative comment.

## Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Quote2
4	4			TPM_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key that can sign the PCR values.
5	20	2S	20	TPM_NONCE	externalData	160 bits of externally supplied data (typically a nonce provided by a server to prevent replay -attacks)
6	<>	3S	<>	TPM_PCR_SELECTION	targetPCR	The indices of the PCRs that are to be reported.
7	1	4S	1	BOOL	addVersion	When TRUE add TPM_CAP_VERSION_INFO to the output
8	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
9	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
10	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
11	20			TPM_AUTHDATA	privAuth	The authorization session digest for inputs and keyHandle. HMAC key: key -> usageAuth.



796 **Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Quote2
4	◇	3S	◇	TPM_PCR_INFO_SHORT	pcrData	The value created and signed for the quote
5	4	4S	4	UINT32	versionInfoSize	Size of the version info
6	◇	5S	◇	TPM_CAP_VERSION_INFO	versionInfo	The version info
7	4	6S	4	UINT32	sigSize	The used size of the output area for the signature
8	◇	7S	◇	BYTE[]	sig	The signed data blob.
9	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
10	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
11	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: Key -> usageAuth.

797 **Actions**

- 798 1. The TPM MUST validate the AuthData to use the key pointed to by keyHandle.
- 799 2. The keyHandle -> sigScheme MUST use SHA-1, return TPM\_INAPPROPRIATE\_SIG if it  
800 does not
- 801 3. Validate targetPCR is a valid TPM\_PCR\_SELECTION structure, on errors return  
802 TPM\_INVALID\_PCR\_INFO
- 803 4. Create H1 a SHA-1 hash of a TPM\_PCR\_COMPOSITE using the TPM\_STCLEAR\_DATA ->  
804 PCR[] indicated by targetPCR -> pcrSelect
- 805 5. Create S1 a TPM\_PCR\_INFO\_SHORT
- 806 a. Set S1->pcrSelection to targetPCR
- 807 b. Set S1->localityAtRelease to TPM\_STANY\_DATA -> localityModifier
- 808 c. Set S1->digestAtRelease to H1
- 809 6. Create Q1 a TPM\_QUOTE\_INFO2 structure
- 810 a. Set Q1 -> fixed to "QUT2"
- 811 b. Set Q1 -> infoShort to S1
- 812 c. Set Q1 -> externalData to externalData
- 813 7. If addVersion is TRUE
- 814 a. Concatenate to Q1 a TPM\_CAP\_VERSION\_INFO structure
- 815 b. Set the output parameters for versionInfo

- 816 8. Else
- 817     a. Set versionInfoSize to 0
- 818     b. Return no bytes in versionInfo
- 819 9. Sign a SHA-1 hash of Q1 using keyHandle as the signature key
- 820 10. Return the signature in sig

## 17. Changing AuthData

### 17.1 TPM\_ChangeAuth

#### Start of informative comment:

The TPM\_ChangeAuth command allows the owner of an entity to change the AuthData for the entity.

TPM\_ChangeAuth requires the encryption of one parameter ("NewAuth"). For the sake of uniformity with other commands that require the encryption of more than one parameter, the parameters used for used encryption are generated from the authLastNonceEven (created during the OSAP session), nonceOdd, and the session shared secret.

The parameter list to this command must always include two authorization sessions, regardless of the state of authDataUsage for the respective keys.

#### End of informative comment.

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ChangeAuth
4	4			TPM_KEY_HANDLE	parentHandle	Handle of the parent key to the entity.
5	2	2S	2	TPM_PROTOCOL_ID	protocolID	The protocol in use.
6	20	3S	20	TPM_ENCAUTH	newAuth	The encrypted new AuthData for the entity. The encryption key is the shared secret from the OSAP protocol.
7	2	4S	2	TPM_ENTITY_TYPE	entityType	The type of entity to be modified
8	4	5S	4	UINT32	encDataSize	The size of the encData parameter
9	<>	6S	<>	BYTE[]	encData	The encrypted entity that is to be modified.
10	4			TPM_AUTHHANDLE	parentAuthHandle	The authorization session handle used for the parent key.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
11	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with parentAuthHandle
12	1	4H1	1	BOOL	continueAuthSession	Ignored, parentAuthHandle is always terminated.
13	20			TPM_AUTHDATA	parentAuth	The authorization session digest for inputs and parentHandle. HMAC key: parentKey.usageAuth.
14	4			TPM_AUTHHANDLE	entityAuthHandle	The authorization session handle used for the encrypted entity. The session type MUST be OIAP
		2H2	20	TPM_NONCE	entitylastNonceEven	Even nonce previously generated by TPM
15	20	3H2	20	TPM_NONCE	entitynonceOdd	Nonce generated by system associated with entityAuthHandle
16	1	4H2	1	BOOL	continueEntitySession	Ignored, entityAuthHandle is always terminated.
17	20			TPM_AUTHDATA	entityAuth	The authorization session digest for the inputs and encrypted entity. HMAC key: entity.usageAuth.

## 833 Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation. See section 4.3.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ChangeAuth
4	4	3S	4	UINT32	outDataSize	The used size of the output area for outData
5	◊	4S	◊	BYTE[]	outData	The modified, encrypted entity.
6	20	2 H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3 H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with parentAuthHandle
7	1	4 H1	1	BOOL	continueAuthSession	Continue use flag, fixed value of FALSE
8	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters and parentHandle. HMAC key: parentKey.usageAuth.
9	20	2 H2	20	TPM_NONCE	entityNonceEven	Even nonce newly generated by TPM to cover entity
		3 H2	20	TPM_NONCE	entitynonceOdd	Nonce generated by system associated with entityAuthHandle
10	1	4 H2	1	BOOL	continueEntity Session	Continue use flag, fixed value of FALSE
11	20			TPM_AUTHDATA	entityAuth	The authorization session digest for the returned parameters and entity. HMAC key: entity.usageAuth, the original and not the new auth value

## 834 Description

1. The parentAuthHandle session type MUST be TPM\_PID\_OSAP.
2. In this capability, the SRK cannot be accessed as entityType TPM\_ET\_KEY, since the SRK is not wrapped by a parent key.

## 838 Actions

1. Verify that entityType is one of TPM\_ET\_DATA, TPM\_ET\_KEY and return the error TPM\_WRONG\_ENTITYTYPE if not.
2. Verify that parentAuthHandle session type is TPM\_PID\_OSAP return TPM\_BAD\_MODE on error
3. Verify that entityAuthHandle session type is TPM\_PID\_OIAP return TPM\_BAD\_MODE on error
4. The encData parameter MUST be the encData field from either the TPM\_STORED\_DATA or TPM\_KEY structures.
5. If parentAuthHandle indicates XOR encryption for the AuthData secrets
  - a. Create X1 the SHA-1 of the concatenation of (parentAuthHandle -> sharedSecret || authLastNonceEven)
  - b. Create decryptAuth by XOR X1 and newAuth
6. Else

- 852       a. Create newAuth by decrypting newAuth using the algorithm indicated in the OSAP  
853       session
- 854       b. Key is from parentAuthHandle -> sharedSecret
- 855       c. IV is SHA-1 of (authLastNonceEven || nonceOdd)
- 856    7. The TPM MUST validate the command using the AuthData in the parentAuth parameter
- 857    8. After parameter validation the TPM creates b1 by decrypting encData using the key  
858       pointed to by parentHandle.
- 859    9. The TPM MUST validate that b1 is a valid TPM structure, either a  
860       TPM\_STORE\_ASYMKEY or a TPM\_SEALED\_DATA
- 861       a. Check the tag, length and authValue for match, return TPM\_INVALID\_STRUCTURE  
862       on any mismatch
- 863    10. The TPM replaces the AuthData for b1 with decryptAuth created above.
- 864    11. The TPM encrypts b1 using the appropriate mechanism for the type using the  
865       parentKeyHandle to provide the key information.
- 866    12. The TPM MUST enforce the destruction of both the parentAuthHandle and  
867       entityAuthHandle sessions.

## 17.2 TPM\_ChangeAuthOwner

### Start of informative comment:

The TPM\_ChangeAuthOwner command allows the owner of an entity to change the AuthData for the TPM Owner or the SRK.

This command requires authorization from the current TPM Owner to execute.

### End of informative comment.

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ChangeAuthOwner
4	2	2S	2	TPM_PROTOCOL_ID	protocolID	The protocol in use.
5	20	3S	20	TPM_ENCAUTH	newAuth	The encrypted new AuthData for the entity. The encryption key is the shared secret from the OSAP protocol.
6	2	4S	2	TPM_ENTITY_TYPE	entityType	The type of entity to be modified
7	4			TPM_AUTHHANDLE	ownerAuthHandle	The authorization session handle used for the TPM Owner.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with ownerAuthHandle
9	1	4H1	1	BOOL	continueAuthSession	Continue use flag the TPM ignores this value
10	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and ownerHandle. HMAC key: ownerAuth.

### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ChangeAuthOwner
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with ownerAuthHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, fixed value of FALSE
6	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters and ownerHandle. HMAC key: ownerAuth, the original value and not the new auth value

876 **Actions**

- 877 1. The TPM MUST validate the command using the AuthData in the ownerAuth parameter
- 878 2. The ownerAuthHandle session type MUST be TPM\_PID\_OSAF
- 879 3. Verify that entityType is either TPM\_ET\_OWNER or TPM\_ET\_SRK, and return the error  
880 TPM\_WRONG\_ENTITYTYPE if not.
- 881 4. If ownerAuthHandle indicates XOR encryption for the AuthData secrets
  - 882 a. Create X1 the SHA-1 of the concatenation of (ownerAuthHandle -> sharedSecret ||  
883 authLastNonceEven)
  - 884 b. Create decryptAuth by XOR X1 and newAuth
- 885 5. Else
  - 886 a. Create newAuth by decrypting newAuth using the algorithm indicated in the OSAF  
887 session
  - 888 b. Key is the previous ownerAuth
  - 889 c. IV is SHA-1 of (authLastNonceEven || nonceOdd)
- 890 6. The TPM MUST enforce the destruction of the ownerAuthHandle session upon  
891 completion of this command (successful or unsuccessful). This includes setting  
892 continueAuthSession to FALSE
- 893 7. Set the AuthData for the indicated entity to decryptAuth
- 894 8. Invalidate all sessions, active or saved

## 18. Authorization Sessions

### 18.1 TPM\_OIAP

#### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OIAP.

#### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OIAP.
4	4			TPM_AUTHHANDLE	authHandle	Handle that TPM creates that points to the authorization state.
5	20			TPM_NONCE	nonceEven	Nonce generated by TPM and associated with session.

#### Actions

1. The TPM\_OIAP command allows the creation of an authorization session handle and the tracking of the handle by the TPM. The TPM generates the handle and nonce.
2. The TPM has an internal limit as to the number of handles that may be open at one time, so the request for a new handle may fail if there is insufficient space available.
3. Internally the TPM will do the following:
  - a. TPM allocates space to save handle, protocol identification, both nonces and any other information the TPM needs to manage the session.
  - b. TPM generates authHandle and nonceEven, returns these to caller
4. On each subsequent use of the OIAP session the TPM MUST generate a new nonceEven value.
5. When TPM\_OIAP is wrapped in an encrypted transport session, no input or output parameters are encrypted.

#### 18.1.1 Actions to validate an OIAP session

##### Start of informative comment:

This section describes the authorization-related actions of a TPM when it receives a command that has been authorized with the OIAP protocol.



Many commands use OIAP authorization. The following description is therefore necessarily abstract.

**End of informative comment.**

## Actions

The TPM MUST perform the following operations:

1. The TPM MUST verify that the authorization session handle (H, say) referenced in the command points to a valid session. If it does not, the TPM returns the error code TPM\_INVALID\_AUTHHANDLE
2. The TPM SHALL retrieve the latest version of the caller's nonce (nonceOdd) and continueAuthSession flag from the input parameter list, and store it in internal TPM memory with the authSession 'H'.
3. The TPM SHALL retrieve the latest version of the TPM's nonce stored with the authorization session H (authLastNonceEven) computed during the previously executed command.
4. The TPM MUST retrieve the secret AuthData (SecretE, say) of the target entity. The entity and its secret must have been previously loaded into the TPM.
  - a. If the command using the OIAP session requires owner authorization
    - i. If TPM\_STCLEAR\_DATA -> ownerReference is TPM\_KH\_OWNER, the secret AuthData is TPM\_PERMANENT\_DATA -> ownerAuth
    - ii. If TPM\_STCLEAR\_DATA -> ownerReference is pointing to a delegate row
      - (1) Set R1 a row index to TPM\_STCLEAR\_DATA -> ownerReference
      - (2) Set D1 a TPM\_DELEGATE\_TABLE\_ROW to TPM\_PERMANENT\_DATA -> delegateTable -> delRow[R1]
      - (3) Set the secret AuthData to D1 -> authValue
      - (4) Validate the TPM\_DELEGATE\_PUBLIC D1 -> pub based on the command ordinal
5. The TPM SHALL perform a HMAC calculation using the entity secret data, ordinal, input command parameters and authorization parameters per Part 1 Object-Independent Authorization Protocol.
6. The TPM SHALL compare HM to the AuthData value received in the input parameters. If they are different, the TPM returns the error code TPM\_AUTHFAIL if the authorization session is the first session of a command, or TPM\_AUTH2FAIL if the authorization session is the second session of a command. Otherwise, the TPM executes the command which (for this example) produces an output that requires authentication.
7. The TPM SHALL generate a nonce (nonceEven).
8. The TPM creates an HMAC digest to authenticate the return code, return values and authorization parameters to the same entity secret per Part 1 Object-Independent Authorization Protocol.
9. The TPM returns the return code, output parameters, authorization parameters and authorization session digest.

956 10.If the output continueUse flag is FALSE, then the TPM SHALL terminate the session.  
957 Future references to H will return an error.

**18.2 TPM\_OSAP****Start of informative comment:**

The TPM\_OSAP command creates the authorization session handle, the shared secret and generates nonceEven and nonceEvenOSAP.

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OSAP.
4	2			TPM_ENTITY_TYPE	entityType	The type of entity in use
5	4			UINT32	entityValue	The selection value based on entityType, e.g. a keyHandle #
6	20			TPM_NONCE	nonceOddOSAP	The nonce generated by the caller associated with the shared secret.

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OSAP.
4	4			TPM_AUTHHANDLE	authHandle	Handle that TPM creates that points to the authorization state.
5	20			TPM_NONCE	nonceEven	Nonce generated by TPM and associated with session.
6	20			TPM_NONCE	nonceEvenOSAP	Nonce generated by TPM and associated with shared secret.

**Description**

1. The TPM\_OSAP command allows the creation of an authorization session handle and the tracking of the handle by the TPM. The TPM generates the handle, nonceEven and nonceEvenOSAP.
2. The TPM has an internal limit on the number of handles that may be open at one time, so the request for a new handle may fail if there is insufficient space available.
3. The TPM\_OSAP allows the binding of an authorization to a specific entity. This allows the caller to continue to send in AuthData for each command but not have to request the information or cache the actual AuthData.
4. When TPM\_OSAP is wrapped in an encrypted transport session, no input or output parameters are encrypted.
5. If the owner pointer is pointing to a delegate row, the TPM internally MUST treat the OSAP session as a DSAP session

978 6. TPM\_ET\_SRK or TPM\_ET\_KEYHANDLE with a value of TPM\_KH\_SRK MUST specify the  
979 SRK.

980 **Actions**

- 981 1. The TPM creates S1 a storage area that keeps track of the information associated with  
982 the authorization.
- 983 2. S1 MUST track the following information
- 984 a. Protocol identification (i.e. TPM\_PID\_OSAP)
- 985 b. nonceEven
- 986 i. Initialized to the next value from the TPM RNG
- 987 c. shared secret
- 988 d. ADIP encryption scheme from TPM\_ENTITY\_TYPE entityType
- 989 e. Any other internal TPM state the TPM needs to manage the session
- 990 3. The TPM MUST create and MAY track the following information
- 991 a. nonceEvenOSAP
- 992 i. Initialized to the next value from the TPM RNG
- 993 4. The TPM calculates the shared secret using an HMAC calculation. The key for the HMAC  
994 calculation is the secret AuthData assigned to the key handle identified by entityValue.  
995 The input to the HMAC calculation is the concatenation of nonces nonceEvenOSAP and  
996 nonceOddOSAP. The output of the HMAC calculation is the shared secret which is saved  
997 in the authorization area associated with authHandle
- 998 5. Check if the ADIP encryption scheme specified by entityType is supported, if not return  
999 TPM\_INAPPROPRIATE\_ENC.
- 000 6. If entityType = TPM\_ET\_KEYHANDLE
- 001 a. The entity to authorize is a key held in the TPM. entityValue contains the keyHandle  
002 that holds the key.
- 003 b. If entityValue is TPM\_KH\_OPERATOR return TPM\_BAD\_HANDLE
- 004 7. else if entityType = TPM\_ET\_OWNER
- 005 a. This value indicates that the entity is the TPM owner. entityValue is ignored
- 006 b. The HMAC key is the secret pointed to by ownerReference (owner secret or delegated  
007 secret)
- 008 8. else if entityType = TPM\_ET\_SRK
- 009 a. The entity to authorize is the SRK. entityValue is ignored.
- 010 9. else if entityType = TPM\_ET\_COUNTER
- 011 a. The entity is a monotonic counter, entityValue contains the counter handle
- 012 10. else if entityType = TPM\_ET\_NV
- 013 a. The entity is a NV index, entityValue contains the NV index
- 014 11. else return TPM\_BAD\_PARAMETER

- 015 12. On each subsequent use of the OSAP session the TPM MUST generate a new nonce  
016 value.
- 017 13. The TPM MUST ensure that OSAP shared secret is only available while the OSAP session  
018 is valid.
- 019 14. The session MUST terminate upon any of the following conditions:
- 020 a. The command that uses the session returns an error
- 021 b. The resource is evicted from the TPM or otherwise invalidated
- 022 c. The session is used in any command for which the shared secret is used to encrypt  
023 an input parameter (TPM\_ENCAUTH)
- 024 d. The TPM Owner is cleared
- 025 e. TPM\_ChangeAuthOwner is executed and this session is attached to the owner  
026 authorization
- 027 f. The session explicitly terminated with continueAuth, TPM\_Reset or  
028 TPM\_FlushSpecific
- 029 g. All OSAP sessions MUST be invalidated when any of the following commands  
030 execute:
- 031 i. TPM\_Delegate\_Manage
- 032 ii. TPM\_Delegate\_CreateOwnerDelegation with Increment==TRUE
- 033 iii. TPM\_Delegate\_LoadOwnerDelegation

## 034 18.2.1 Actions to validate an OSAP session

### 035 **Start of informative comment:**

036 This section describes the authorization-related actions of a TPM when it receives a  
037 command that has been authorized with the OSAP protocol.

038 Many commands use OSAP authorization. The following description is therefore necessarily  
039 abstract.

### 040 **End of informative comment**

## 041 **Actions**

- 042 1. On reception of a command with ordinal C1 that uses an authorization session, the TPM  
043 SHALL perform the following actions:
- 044 2. The TPM MUST have been able to retrieve the shared secret (Shared, say) of the target  
045 entity when the authorization session was established with TPM\_OSAP. The entity and  
046 its secret must have been previously loaded into the TPM.
- 047 3. The TPM MUST verify that the authorization session handle (H, say) referenced in the  
048 command points to a valid session. If it does not, the TPM returns the error code  
049 TPM\_INVALID\_AUTHHANDLE.
- 050 4. The TPM MUST calculate the HMAC (HM1, say) of the command parameters according  
051 to Part 1 Object-Specific Authorization Protocol.

- 052 5. The TPM SHALL compare HM1 to the AuthData value received in the command. If they  
053 are different, the TPM returns the error code TPM\_AUTHFAIL if the authorization session  
054 is the first session of a command, or TPM\_AUTH2FAIL if the authorization session is the  
055 second session of a command., the TPM executes command C1 which produces an  
056 output (O, say) that requires authentication and uses a particular return code (RC, say).
- 057 6. The TPM SHALL generate the latest version of the even nonce (nonceEven).
- 058 7. The TPM MUST calculate the HMAC (HM2) of the return parameters according to section  
059 Part 1 Object-Specific Authorization Protocol.
- 060 8. The TPM returns HM2 in the parameter list.
- 061 9. The TPM SHALL retrieve the continue flag from the received command. If the flag is  
062 FALSE, the TPM SHALL terminate the session and destroy the thread associated with  
063 handle H.
- 064 10.If the shared secret was used to provide confidentiality for data in the received  
065 command, the TPM SHALL terminate the session and destroy the thread associated with  
066 handle H.
- 067 11.Each time that access to an entity (key) is authorized using OSAP, the TPM MUST  
068 ensure that the OSAP shared secret is that derived from the entity using TPM\_OSAP

**18.3 TPM\_DSAP****Start of informative comment:**

The TPM\_DSAP command creates the authorization session handle using a delegated AuthData value passed into the command as an encrypted blob or from the internal delegation table. It can be used to start an authorization session for a user key or the owner.

Identically to TPM\_OSAP, it generates a shared secret and generates nonceEven and nonceEvenOSAP.

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DSAP.
4	2			TPM_ENTITY_TYPE	entityType	The type of delegation information to use
5	4			TPM_KEY_HANDLE	keyHandle	Key for which delegated authority corresponds, or 0 if delegated owner activity. Only relevant if entityValue equals TPM_DELEGATE_KEY_BLOB
6	20			TPM_NONCE	nonceOddDSAP	The nonce generated by the caller associated with the shared secret.
7	4			UINT32	entityValueSize	The size of entityValue.
8	<>	2S	<>	BYTE [ ]	entityValue	TPM_DELEGATE_KEY_BLOB or TPM_DELEGATE_OWNER_BLOB or index MUST not be empty If entityType is TPM_ET_DEL_ROW then entityValue is a TPM_DELEGATE_INDEX

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DSAP.
4	4			TPM_AUTHHANDLE	authHandle	Handle that TPM creates that points to the authorization state.
5	20			TPM_NONCE	nonceEven	Nonce generated by TPM and associated with session.
6	20			TPM_NONCE	nonceEvenDSAP	Nonce generated by TPM and associated with shared secret.

**Description**

1. The TPM\_DSAP command allows the creation of an authorization session handle and the tracking of the handle by the TPM. The TPM generates the handle, nonceEven and nonceEvenOSAP.

2. The TPM has an internal limit on the number of handles that may be open at one time, so the request for a new handle may fail if there is insufficient space available.
  3. The TPM\_DSAP allows the binding of a delegated authorization to a specific entity. This allows the caller to continue to send in AuthData for each command but not have to request the information or cache the actual AuthData.
  4. Each ordinal that uses the DSAP session MUST validate that TPM\_PERMANENT\_DATA -> restrictDelegate does not restrict delegation, based on keyHandle -> keyUsage and keyHandle -> keyFlags, return TPM\_INVALID\_KEYUSAGE on error.
  5. On each subsequent use of the DSAP session the TPM MUST generate a new nonce value and check if the ordinal to be executed has delegation to execute. The TPM MUST ensure that the DSAP shared secret is only available while the DSAP session is valid.
  6. When TPM\_DSAP is wrapped in an encrypted transport session
    - a. For input the only parameter encrypted is entityValue
    - b. For output no parameters are encrypted
  7. The DSAP session MUST terminate under any of the following conditions
    - a. The command that uses the session returns an error
    - b. If attached to a key, when the key is evicted from the TPM or otherwise invalidated
    - c. The session is used in any command for which the shared secret is used to encrypt an input parameter (TPM\_ENCAUTH)
    - d. The TPM Owner is cleared
    - e. TPM\_ChangeAuthOwner is executed and this session is attached to the owner authorization
    - f. The session explicitly terminated with continueAuth, TPM\_Reset or TPM\_FlushSpecific
    - g. All DSAP sessions MUST be invalidated when any of the following commands execute:
      - i. TPM\_Delegate\_CreateOwnerDelegation
      - ii. When Increment is TRUE
      - iii. TPM\_Delegate\_LoadOwnerDelegation
      - iv. TPM\_Delegate\_Manage
- entityType = TPM\_ET\_DEL\_OWNER\_BLOB
- The entityValue parameter contains an owner delegation blob structure.
- entityType = TPM\_ET\_DEL\_ROW
- The entityValue parameter contains a row number in the nv Delegation table which should be used for the AuthData value.
- entityType = TPM\_DEL\_KEY\_BLOB
- The entityValue parameter contains a key delegation blob structure.

## Actions



```
122 1. If entityType == TPM_ET_DEL_OWNER_BLOB
123   a. Map entityValue to B1 a TPM_DELEGATE_OWNER_BLOB
124   b. Validate that B1 is a valid TPM_DELEGATE_OWNER_BLOB, return
125      TPM_WRONG_ENTITYTYPE on error
126   c. Locate B1 -> pub -> familyID in the TPM_FAMILY_TABLE and set familyRow to
127      indicate row, return TPM_BADINDEX if not found
128   d. Set FR to TPM_FAMILY_TABLE.famTableRow[familyRow]
129   e. If FR -> flags TPM_FAMFLAG_ENABLED is FALSE, return TPM_DISABLED_CMD
130   f. Verify that B1->verificationCount equals FR -> verificationCount.
131   g. Validate the integrity of the blob
132      i. Copy B1 -> integrityDigest to H2
133      ii. Set B1 -> integrityDigest to NULL
134      iii. Create H3 the HMAC of B1 using tpmProof as the secret
135      iv. Compare H2 to H3 return TPM_AUTHFAIL on mismatch
136   h. Create S1 a TPM_DELEGATE_SENSITIVE by decrypting B1 -> sensitiveArea using
137      TPM_DELEGATE_KEY
138   i. Validate S1 values
139      i. S1 -> tag is TPM_TAG_DELEGATE_SENSITIVE
140      ii. Return TPM_BAD_DELEGATE on error
141   j. Set A1 to S1 -> authValue
142 2. Else if entityType == TPM_ET_DEL_ROW
143   a. Verify that entityValue points to a valid row in the delegation table.
144   b. Set D1 to the delegation information in the row.
145   c. Set A1 to D1->authValue.
146   d. Locate D1 -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate that
147      row, return TPM_BADINDEX if not found
148   e. Set FR to TPM_FAMILY_TABLE.famTableRow[familyRow]
149   f. If FR -> flags TPM_FAMFLAG_ENABLED is FALSE, return TPM_DISABLED_CMD
150   g. Verify that D1->verificationCount equals FR -> verificationCount.
151 3. Else if entityType == TPM_ET_DEL_KEY_BLOB
152   a. Map entityValue to K1 a TPM_DELEGATE_KEY_BLOB
153   b. Validate that K1 is a valid TPM_DELEGATE_KEY_BLOB, return
154      TPM_WRONG_ENTITYTYPE on error
155   c. Locate K1 -> pub -> familyID in the TPM_FAMILY_TABLE and set familyRow to
156      indicate that row, return TPM_BADINDEX if not found
157   d. Set FR to TPM_FAMILY_TABLE.famTableRow[familyRow]
```

- 158 e. If FR -> flags TPM\_FAMFLAG\_ENABLED is FALSE, return TPM\_DISABLED\_CMD
- 159 f. Verify that K1 -> pub -> verificationCount equals FR -> verificationCount.
- 160 g. Validate the integrity of the blob
  - 161 i. Copy K1 -> integrityDigest to H2
  - 162 ii. Set K1 -> integrityDigest to NULL
  - 163 iii. Create H3 the HMAC of K1 using tpmProof as the secret
  - 164 iv. Compare H2 to H3 return TPM\_AUTHFAIL on mismatch
- 165 h. Validate that K1 -> pubKeyDigest identifies keyHandle, return TPM\_KEYNOTFOUND
- 166 on error
- 167 i. Create S1 a TPM\_DELEGATE\_SENSITIVE by decrypting K1 -> sensitiveArea using
- 168 TPM\_DELEGATE\_KEY
- 169 j. Validate S1 values
  - 170 i. S1 -> tag is TPM\_TAG\_DELEGATE\_SENSITIVE
  - 171 ii. Return TPM\_BAD\_DELEGATE on error
- 172 k. Set A1 to S1 -> authValue
- 173 4. Else return TPM\_BAD\_PARAMETER
- 174 5. Generate a new authorization session handle and reserve space to save protocol
- 175 identification, shared secret, pcrInfo, both nonces, ADIP encryption scheme, delegated
- 176 permission bits and any other information the TPM needs to manage the session.
- 177 6. Read two new values from the RNG to generate nonceEven and nonceEvenOSAP.
- 178 7. The TPM calculates the shared secret using an HMAC calculation. The key for the HMAC
- 179 calculation is A1. The input to the HMAC calculation is the concatenation of nonces
- 180 nonceEvenOSAP and nonceOddOSAP. The output of the HMAC calculation is the shared
- 181 secret which is saved in the authorization area associated with authHandle.

**18.4 TPM\_SetOwnerPointer****Start of informative comment:**

This command will set a reference to which secret the TPM will use when executing an owner secret related OIAP or OSAP session.

This command should only be used to provide an owner delegation function for legacy code that does not itself support delegation. Normally, TPM\_STCLEAR\_DATA->ownerReference points to TPM\_KH\_OWNER, indicating that OIAP and OSAP sessions should use the owner authorization. This command allows ownerReference to point to an index in the delegation table, indicating that OIAP and OSAP sessions should use the delegation authorization.

In use, a TSS supporting delegation would create and load the owner delegation and set the owner pointer to that delegation. From then on, a legacy TSS application would use its OIAP and OSAP sessions with the delegated owner authorization.

Since this command is not authorized, the ownerReference is open to DoS attacks. Applications can attempt to recover from a failing owner authorization by resetting ownerReference to an appropriate value.

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Ordinal: TPM_ORD_SetOwnerPointer
4	2	2S	2	TPM_ENTITY_TYPE	entityType	The type of entity in use
5	4	3S	4	UINT32	entityValue	The selection value based on entityType

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	Ordinal: TPM_ORD_SetOwnerPointer

**Actions**

1. Map TPM\_STCLEAR\_DATA to V1
2. If entityType = TPM\_ET\_DEL\_ROW
  - a. This value indicates that the entity is a delegate row. entityValue is a delegate index in the delegation table.

- 205        b. Validate that entityValue points to a legal row within the delegate table stored within  
206        the TPM. If not return TPM\_BADINDEX
- 207           i. Set D1 to the delegation information in the row.
- 208        c. Locate D1 -> familyID in the TPM\_FAMILY\_TABLE and set familyRow to indicate that  
209        row, return TPM\_BADINDEX if not found.
- 210        d. Set FR to TPM\_FAMILY\_TABLE.famTableRow[familyRow]
- 211        e. If FR -> flags TPM\_FAMFLAG\_ENABLED is FALSE, return TPM\_DISABLED\_CMD
- 212        f. Verify that B1->verificationCount equals FR -> verificationCount.
- 213        g. The TPM sets V1-> ownerReference to entityValue
- 214        h. Return TPM\_SUCCESS
- 215    3. else if entityType = TPM\_ET\_OWNER
- 216        a. This value indicates that the entity is the TPM owner. entityValue is ignored.
- 217        b. The TPM sets V1-> ownerReference to TPM\_KH\_OWNER
- 218        c. Return TPM\_SUCCESS
- 219    4. Return TPM\_BAD\_PARAMETER

## 19. Delegation Commands

### 19.1 TPM\_Delegate\_Manage

**Start of informative comment:**

TPM\_Delegate\_Manage is the fundamental process for managing the Family tables, including enabling/disabling Delegation for a selected Family. Normally TPM\_Delegate\_Manage must be executed at least once (to create Family tables for a particular family) before any other type of Delegation command in that family can succeed.

TPM\_Delegate\_Manage is authorized by the TPM Owner if an Owner is installed, because changing a table is a privileged Owner operation. If no Owner is installed, TPM\_Delegate\_Manage requires no privilege to execute. This does not disenfranchise an Owner, since there is no Owner, and simplifies loading of tables during platform manufacture or on first-boot. Burn-out of TPM non-volatile storage by inappropriate use is mitigated by the TPM's normal limits on NV-writes in the absence of an Owner. Tables can be locked after loading, to prevent subsequent tampering, and only unlocked by the Owner, his delegate, or the act of removing the Owner (even if there is no Owner).

TPM\_Delegate\_Manage command is customized by opCode:

(1) TPM\_FAMILY\_ENABLE enables/disables use of a family and all the rows of the delegate table belonging to that family,

(2) TPM\_FAMILY\_ADMIN can be used to prevent further management of the Tables until an Owner is installed, or until the Owner is removed from the TPM. (Note that the Physical Presence command TPM\_ForceClear always enables further management, even if TPM\_ForceClear is used when no Owner is installed.)

(3) TPM\_FAMILY\_CREATE creates a new family. Sessions are invalidated even in this case because the lastFamilyID could wrap.

(4) TPM\_FAMILY\_INVALIDATE invalidates an existing family.

**End of informative comment.**

## 246 Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Delegate_Manage
4	4	2S	4	TPM_FAMILY_ID	familyID	The familyID that is to be managed
5	4	3s	4	TPM_FAMILY_OPERATION	opCode	Operation to be performed by this command.
6	4	4s	4	UINT32	opDataSize	Size in bytes of opData
7	<	5s	<	BYTE [ ]	opData	Data necessary to implement opCode
8	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
9	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
10	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
11	20			TPM_AUTHDATA	ownerAuth	HMAC key: ownerAuth.

## 247 Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Delegate_Manage
4	4	3S	4	UINT32	retDataSize	Size in bytes of retData
5	<	4S	<	BYTE [ ]	retData	Returned data
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20			TPM_AUTHDATA	resAuth	HMAC key: ownerAuth.

## 248 Action

- 249 1. If opCode != TPM\_FAMILY\_CREATE
- 250 a. Locate familyID in the TPM\_FAMILY\_TABLE and set familyRow to indicate row,  
251 return TPM\_BADINDEX if not found
- 252 b. Set FR, a TPM\_FAMILY\_TABLE\_ENTRY, to TPM\_FAMILY\_TABLE. famTableRow  
253 [familyRow]
- 254 2. If tag = TPM\_TAG\_RQU\_AUTH1\_COMMAND
- 255 a. Validate the command and parameters using ownerAuth, return TPM\_AUTHFAIL on  
256 error

```

257     b. If the authHandle session type is TPM_PID_DSAP
258         i. If opCode = TPM_FAMILY_CREATE
259             (1) The TPM MUST ignore familyID
260         ii. Else
261             (1) Verify that the familyID associated with authHandle matches the familyID
262                 parameter, return TPM_DELEGATE_FAMILY on error
263     3. Else
264         a. If TPM_PERMANENT_DATA -> ownerAuth is valid, return TPM_AUTHFAIL
265         b. If opCode != TPM_FAMILY_CREATE and FR -> flags ->
266             TPM_DELEGATE_ADMIN_LOCK is TRUE, return TPM_DELEGATE_LOCK
267         c. Validate max NV writes without an owner
268             i. Set NV1 to TPM_PERMANENT_DATA -> noOwnerNVWrite
269             ii. Increment NV1 by 1
270             iii. If NV1 > TPM_MAX_NV_WRITE_NOOWNER return TPM_MAXNVWRITES
271             iv. Set TPM_PERMANENT_DATA -> noOwnerNVWrite to NV1
272     4. The TPM invalidates sessions
273         a. MUST invalidate all DSAP sessions
274         b. MUST invalidate all OSAP sessions associated with the delegation table
275         c. MUST set TPM_STCLEAR_DATA -> ownerReference to TPM_KH_OWNER
276         d. MAY invalidate any other session
277     5. If opCode == TPM_FAMILY_CREATE
278         a. Validate that sufficient space exists within the TPM to store an additional family and
279             map F2 to the newly allocated space.
280         b. Validate that opData is a TPM_FAMILY_LABEL
281             i. If opDataSize != sizeof(TPM_FAMILY_LABEL) return TPM_BAD_PARAM_SIZE
282         c. Map F2 to a TPM_FAMILY_TABLE_ENTRY
283             i. Set F2 -> tag to TPM_TAG_FAMILY_TABLE_ENTRY
284             ii. Set F2 -> familyLabel to opData
285         d. Increment TPM_PERMANENT_DATA -> lastFamilyID by 1
286         e. Set F2 -> familyID = TPM_PERMANENT_DATA -> lastFamilyID
287         f. Set F2 -> verificationCount = 1
288         g. Set F2 -> flags -> TPM_FAMFLAG_ENABLED to FALSE
289         h. Set F2 -> flags -> TPM_DELEGATE_ADMIN_LOCK to FALSE
290         i. Set retDataSize = 4
291         j. Set retData = F2 -> familyID

```

292 k. Return TPM\_SUCCESS  
293 6. If authHandle is of type DSAP then continueAuthSession MUST set to FALSE  
294 7. If opCode == TPM\_FAMILY\_ADMIN  
295 a. Validate that opDataSize == 1, and that opData is a Boolean value.  
296 b. Set (FR -> flags -> TPM\_DELEGATE\_ADMIN\_LOCK) = opData  
297 c. Set retDataSize = 0  
298 d. Return TPM\_SUCCESS  
299 8. else If opCode == TPM\_FAMILY\_ENABLE  
300 a. Validate that opDataSize == 1, and that opData is a Boolean value.  
301 b. Set FR -> flags-> TPM\_FAMFLAG\_ENABLED = opData  
302 c. Set retDataSize = 0  
303 d. Return TPM\_SUCCESS  
304 9. else If opCode == TPM\_FAMILY\_INVALIDATE  
305 a. Invalidate all data associated with familyRow  
306 i. All data is all information pointed to by FR  
307 ii. return TPM\_SELFTEST\_FAILED on failure  
308 b. Set retDataSize = 0  
309 c. Return TPM\_SUCCESS  
310 10. Else return TPM\_BAD\_PARAMETER



**19.2 TPM\_Delegate\_CreateKeyDelegation****Start of informative comment:**

This command delegates privilege to use a key by creating a blob that can be used by TPM\_DSAP.

There is no check for appropriateness of the key's key usage against the key permission settings. If the key usage is incorrect, this command succeeds, but the delegated command will fail.

These blobs CANNOT be used as input data for TPM\_LoadOwnerDelegation because the internal TPM delegate table can store owner delegations only.

(TPM\_Delegate\_CreateOwnerDelegation must be used to delegate Owner privilege.)

**End of informative comment****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Delegate_CreateKeyDelegation.
4	4			TPM_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key.
5	◁	2S	◁	TPM_DELEGATE_PUBLIC	publicInfo	The public information necessary to fill in the blob
6	20	3S	20	TPM_ENCAUTH	delAuth	The encrypted new AuthData for the blob. The encryption key is the shared secret from the OSAP protocol.
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	Ignored
10	20			TPM_AUTHDATA	privAuth	The authorization session digest that authorizes the use of keyHandle. HMAC key: key.usageAuth

## 324 Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Delegate_CreateKeyDelegation
4	4	3S	4	UINT32	blobSize	The length of the returned blob
5	<>	4S	<>	TPM_DELEGATE_KEY_BLOB	blob	The partially encrypted delegation information.
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag. Fixed value of FALSE
8	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: key.usageAuth

## 325 Description

1. The use restrictions that may be present on the key pointed to by keyHandle are not enforced for this command. Stated another way, TPM\_CreateKeyDelegation is not a use of the key.

## 329 Action

1. Verify AuthData for the command and parameters using privAuth
2. Locate publicInfo -> familyID in the TPM\_FAMILY\_TABLE and set familyRow to indicate row, return TPM\_BADINDEX if not found
3. If the key authentication is in fact a delegation, then the TPM SHALL validate the command and parameters using Delegation authorisation, then
  - a. Validate that authHandle -> familyID equals publicInfo -> familyID return TPM\_DELEGATE\_FAMILY on error
  - b. If TPM\_FAMILY\_TABLE.famTableRow[ authHandle -> familyID] -> flags -> TPM\_FAMFLAG\_ENABLED is FALSE, return error TPM\_DISABLED\_CMD.
  - c. Verify that the delegation bits in publicInfo do not grant more permissions than currently delegated. Otherwise return error TPM\_AUTHFAIL
4. Check that publicInfo -> delegateType is TPM\_DEL\_KEY\_BITS
5. Verify that authHandle indicates an OSAP or DSAP session return TPM\_INVALID\_AUTHHANDLE on error
6. If authHandle indicates XOR encryption for the AuthData secrets
  - a. Create X1 the SHA-1 of the concatenation of (authHandle -> sharedSecret || authLastNonceEven)
  - b. Create a1 by XOR X1 and delAuth
7. Else

- 349       a. Create a1 by decrypting delAuth using the algorithm indicated in the OSAP session
- 350       b. Key is from authHandle -> sharedSecret
- 351       c. IV is SHA-1 of (authLastNonceEven || nonceOdd)
- 352   8. Create h1 the SHA-1 of TPM\_STORE\_PUBKEY structure of the key pointed to by
- 353       keyHandle
- 354   9. Create M1 a TPM\_DELEGATE\_SENSITIVE structure
- 355       a. Set M1 -> tag to TPM\_TAG\_DELEGATE\_SENSITIVE
- 356       b. Set M1 -> authValue to a1
- 357       c. The TPM MAY add additional information of a sensitive nature relative to the
- 358       delegation
- 359   10. Create M2 the encryption of M1 using TPM\_DELEGATE\_KEY
- 360   11. Create P1 a TPM\_DELEGATE\_KEY\_BLOB
- 361       a. Set P1 -> tag to TPM\_TAG\_DELG\_KEY\_BLOB
- 362       b. Set P1 -> pubKeyDigest to H1
- 363       c. Set P1 -> pub to PublicInfo
- 364       d. Set P1 -> pub -> verificationCount to familyRow -> verificationCount
- 365       e. Set P1 -> integrityDigest to NULL
- 366       f. The TPM sets additionalArea and additionalAreaSize appropriate for this TPM. The
- 367       information MAY include symmetric IV, symmetric mode of encryption and other data
- 368       that allows the TPM to process the blob in the future.
- 369       g. Set P1 -> sensitiveSize to the size of M2
- 370       h. Set P1 -> sensitiveArea to M2
- 371   12. Calculate H2 the HMAC of P1 using tpmProof as the secret
- 372   13. Set P1 -> integrityDigest to H2
- 373   14. Ignore continueAuthSession on input set continueAuthSession to FALSE on output
- 374   15. Return P1 as blob

### 19.3 TPM\_Delegate\_CreateOwnerDelegation

#### Start of informative comment:

TPM\_Delegate\_CreateOwnerDelegation delegates the Owner's privilege to use a set of command ordinals, by creating a blob. Such blobs can be used as input data for TPM\_DSAP or TPM\_Delegate\_LoadOwnerDelegation.

TPM\_Delegate\_CreateOwnerDelegation includes the ability to void all existing delegations (by incrementing the verification count) before creating the new delegation. This ensures that the new delegation will be the only delegation that can operate at Owner privilege in this family. This new delegation could be used to enable a security monitor (a local separate entity, or remote separate entity, or local host entity) to reinitialize a family and perhaps perform external verification of delegation settings. Normally the ordinals for a delegated security monitor would include TPM\_Delegate\_CreateOwnerDelegation (this command) in order to permit the monitor to create further delegations, and TPM\_Delegate\_UpdateVerification to reactivate some previously voided delegations.

If the verification count is incremented and the new delegation does not delegate any privileges (to any ordinals) at all, or uses an authorisation value that is then discarded, this family's delegations are all void and delegation must be managed using actual Owner authorisation.

(TPM\_Delegate\_CreateKeyDelegation must be used to delegate privilege to use a key.)

#### End of informative comment.

#### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	TPM_ORD_Delegate_CreateOwnerDelegation.
4	1	2S	1	BOOL	increment	Flag dictates whether verificationCount will be incremented
5	<>	3S	<>	TPM_DELEGATE_PUBLIC	publicInfo	The public parameters for the blob
6	20	4S	20	TPM_ENCAUTH	delAuth	The encrypted new AuthData for the blob. The encryption key is the shared secret from the OSAP protocol.
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle TPM Owner authentication
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	Ignored
10	20			TPM_AUTHDATA	ownerAuth	The authorization session digest. HMAC key:ownerAuth

396 **Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	TPM_ORD_Delegate_CreateOwnerDelegation
4	4	3S	4	UINT32	blobSize	The length of the returned blob
5	◇	4S	◇	TPM_DELEGATE_OWNER_BLOB	blob	The partially encrypted delegation information.
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag. Fixed value of FALSE
8	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth

397 **Action**

- 398 1. The TPM SHALL authenticate the command using TPM Owner authentication. Return  
399 TPM\_AUTHFAIL on failure.
- 400 2. Locate publicInfo -> familyID in the TPM\_FAMILY\_TABLE and set familyRow to indicate  
401 the row return TPM\_BADINDEX if not found
- 402 a. Set FR to TPM\_FAMILY\_TABLE.famTableRow[familyRow]
- 403 3. If the TPM Owner authentication is in fact a delegation
- 404 a. Validate that authHandle -> familyID equals publicInfo -> familyID return  
405 TPM\_DELEGATE\_FAMILY on error
- 406 b. If FR -> flags -> TPM\_FAMFLAG\_ENABLED is FALSE, return error  
407 TPM\_DISABLED\_CMD.
- 408 c. Verify that the delegation bits in publicInfo do not grant more permissions then  
409 currently delegated. Otherwise, return error TPM\_AUTHFAIL.
- 410 4. Check that publicInfo -> delegateType is TPM\_DEL\_OWNER\_BITS
- 411 5. Verify that authHandle indicates an OSAP or DSAP session return  
412 TPM\_INVALID\_AUTHHANDLE on error
- 413 6. If increment == TRUE
- 414 a. Increment FR -> verificationCount
- 415 b. Set TPM\_STCLEAR\_DATA-> ownerReference to TPM\_KH\_OWNER
- 416 c. The TPM invalidates sessions
- 417 i. MUST invalidate all DSAP sessions
- 418 ii. MUST invalidate all OSAP sessions associated with the delegation table

- 419           iii. MAY invalidate any other session
- 420   7. If authHandle indicates XOR encryption for the AuthData secrets
- 421       a. Create X1 the SHA-1 of the concatenation of (authHandle -> sharedSecret ||
- 422       authLastNonceEven)
- 423       b. Create a1 by XOR X1 and delAuth
- 424   8. Else
- 425       a. Create a1 by decrypting delAuth using the algorithm indicated in the OSAP session
- 426       b. Key is from authHandle -> sharedSecret
- 427       c. IV is SHA-1 of (authLastNonceEven || nonceOdd)
- 428   9. Create M1 a TPM\_DELEGATE\_SENSITIVE structure
- 429       a. Set M1 -> tag to TPM\_TAG\_DELEGATE\_SENSITIVE
- 430       b. Set M1 -> authValue to a1
- 431       c. Set other M1 fields as determined by the TPM vendor
- 432   10. Create M2 the encryption of M1 using TPM\_DELEGATE\_KEY
- 433   11. Create B1 a TPM\_DELEGATE\_OWNER\_BLOB
- 434       a. Set B1 -> tag to TPM\_TAG\_DELG\_OWNER\_BLOB
- 435       b. Set B1 -> pub to publicInfo
- 436       c. Set B1 -> sensitiveSize to the size of M2
- 437       d. Set B1 -> sensitiveArea to M2
- 438       e. Set B1 -> integrityDigest to NULL
- 439       f. Set B1 -> pub -> verificationCount to FR -> verificationCount
- 440   12. The TPM sets additionalArea and additionalAreaSize appropriate for this TPM. The
- 441       information MAY include symmetric IV, symmetric mode of encryption and other data
- 442       that allows the TPM to process the blob in the future.
- 443   13. Create H1 the HMAC of B1 using tpmProof as the secret
- 444   14. Set B1 -> integrityDigest to H1
- 445   15. Ignore continueAuthSession on input set continueAuthSession to FALSE on output
- 446   16. Return B1 as blob

## 19.4 TPM\_Delegate\_LoadOwnerDelegation

### Start of informative comment:

This command loads a delegate table row blob into a non-volatile delegate table row. TPM\_Delegate\_LoadOwnerDelegation can be used during manufacturing or on first boot (when no Owner is installed), or after an Owner is installed. If an Owner is installed, TPM\_Delegate\_LoadOwnerDelegation requires Owner authorisation, and sensitive information must be encrypted.

Burn-out of TPM non-volatile storage by inappropriate use is mitigated by the TPM's normal limits on NV-writes in the absence of an Owner. Tables can be locked after loading using TPM\_Delegate\_Manage, to prevent subsequent tampering.

A management system outside the TPM is expected to manage the delegate table rows stored on the TPM, and can overwrite any previously stored data.

This command cannot be used to load key delegation blobs into the TPM

### End of informative comment.

## Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes incl. paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Ordinal: TPM_ORD_Delegate_LoadOwnerDelegation
4	4	3S	4	TPM_DELEGATE_INDEX	index	The index of the delegate row to be written
5	4	4S	4	UINT32	blobSize	The size of the delegate blob
6	<>	5S	<>	TPM_DELEGATE_OWNER_BLOB	blob	Delegation information, including encrypted portions as appropriate
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle TPM Owner authentication
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
10	20			TPM_AUTHDATA	ownerAuth	The authorization session digest. HMAC key:ownerAuth

## 462 Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	TPM_ORD_Delegate_LoadOwnerDelegation
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	resAuth	Authorization HMAC key: ownerAuth.

## 463 Actions

- 464 1. Map blob to D1 a TPM\_DELEGATE\_OWNER\_BLOB.
- 465 a. Validate that D1 -> tag == TPM\_TAG\_DELEGATE\_OWNER\_BLOB
- 466 2. Locate D1 -> pub -> familyID in the TPM\_FAMILY\_TABLE and set familyRow to indicate
- 467 row, return TPM\_BADINDEX if not found
- 468 3. Set FR to TPM\_FAMILY\_TABLE -> famTableRow[familyRow]
- 469 4. If TPM Owner is installed
- 470 a. Validate the command and parameters using TPM Owner authentication, return
- 471 TPM\_AUTHFAIL on error
- 472 b. If the authHandle session type is TPM\_PID\_DSAP, verify that D1 -> pub -> familyID
- 473 matches authHandle -> familyID, on error return TPM\_DELEGATE\_FAMILY
- 474 5. Else
- 475 a. If FR -> flags -> TPM\_DELEGATE\_ADMIN\_LOCK is TRUE return
- 476 TPM\_DELEGATE\_LOCK
- 477 b. Validate max NV writes without an owner
- 478 i. Set NV1 to PD -> noOwnerNVWrite
- 479 ii. Increment NV1 by 1
- 480 iii. If NV1 > TPM\_MAX\_NV\_WRITE\_NOOWNER return TPM\_MAXNVWRITES
- 481 iv. Set PD -> noOwnerNVWrite to NV1
- 482 6. If FR -> flags -> TPM\_FAMFLAG\_ENABLED is FALSE, return TPM\_DISABLED\_CMD
- 483 7. If TPM Owner is installed, validate the integrity of the blob
- 484 a. Copy D1 -> integrityDigest to H2
- 485 b. Set D1 -> integrityDigest to NULL
- 486 c. Create H3 the HMAC of D1 using tpmProof as the secret



487       d. Compare H2 to H3, return TPM\_AUTHFAIL on mismatch

488   8. If TPM Owner is installed, create S1 a TPM\_DELEGATE\_SENSITIVE area by decrypting

489     D1 -> sensitiveArea using TPM\_DELEGATE\_KEY. Otherwise set S1 = D1 -> sensitiveArea

490   9. Validate S1

491     a. Validate that S1 -> tag == TPM\_TAG\_DELEGATE\_SENSITIVE, return

492     TPM\_INVALID\_STRUCTURE on error

493   10. Validate that index is a valid value for delegateTable, return TPM\_BADINDEX on error

494   11. The TPM invalidates sessions

495     a. MUST invalidate all DSAP sessions

496     b. MUST invalidate all OSAP sessions associated with the delegation table

497     c. MAY invalidate any other session

498   12. Copy data to the delegate table row

499     a. Copy the TPM\_DELEGATE\_PUBLIC from D1 -> pub to TPM\_DELEGATE\_TABLE ->

500     delRow[index] -> pub.

501     b. Copy the TPM\_SECRET from S1 -> authValue to TPM\_DELEGATE\_TABLE ->

502     delRow[index] -> authValue.

503     c. Set TPM\_STCLEAR\_DATA -> ownerReference to TPM\_KH\_OWNER

504     d. If authHandle is of type DSAP then continueAuthSession MUST set to FALSE

505   13. Return TPM\_SUCCESS

## 19.5 TPM\_Delegate\_ReadTable

### Start of informative comment:

This command reads from the TPM the public contents of the family and delegate tables that are stored on the TPM. Such data is required during external verification of tables.

There are no restrictions on the execution of this command; anyone can read this information regardless of the state of the PCRs, regardless of whether they know any specific AuthData value and regardless of whether or not the enable and admin bits are set one way or the other.

### End of informative comment.

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_Delegate_ReadTable

### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_Delegate_ReadTable
4	4	3S	4	UINT32	familyTableSize	Size in bytes of familyTable
5	<>	4S	<>	BYTE [ ]	familyTable	Array of TPM_FAMILY_TABLE_ENTRY elements
6	4	5S	4	UINT32	delegateTableSize	Size in bytes of delegateTable
7	<>	6S	<>	BYTE[]	delegateTable	Array of TPM_DELEGATE_INDEX and TPM_DELEGATE_PUBLIC elements

### Actions

1. Set familyTableSize to the number of valid families on the TPM times sizeof(TPM\_FAMILY\_TABLE\_ELEMENT).
2. Copy the valid entries in the internal family table to the output array familyTable
3. Set delegateTableSize to the number of valid delegate table entries on the TPM times (sizeof(TPM\_DELEGATE\_PUBLIC) + 4).
4. For each valid entry
  - a. Write the TPM\_DELEGATE\_INDEX to delegateTable
  - b. Copy the TPM\_DELEGATE\_PUBLIC to delegateTable

526 5. Return TPM\_SUCCESS

## 19.6 TPM\_Delegate\_UpdateVerification

### Start of informative comment:

TPM\_UpdateVerification sets the verificationCount in an entity (a blob or a delegation row) to the current family value, in order that the delegations represented by that entity will continue to be accepted by the TPM.

### End of informative comment.

## Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Delegate_UpdateVerification
4	4	2S	4	UINT32	inputSize	The size of inputData
5	◇	3S	◇	BYTE	inputData	TPM_DELEGATE_KEY_BLOB or TPM_DELEGATE_OWNER_BLOB or TPM_DELEGATE_INDEX
6	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
7	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
9	20			TPM_AUTHDATA	ownerAuth	Authorization HMAC key: ownerAuth.

534 **Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Delegate_UpdateVerification
4	4	3S	4	UINT32	outputSize	The size of the output
5	◁	4S	◁	BYTE	outputData	TPM_DELEGATE_KEY_BLOB or TPM_DELEGATE_OWNER_BLOB
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

535 **Actions**

- 536 1. Verify the TPM Owner, directly or indirectly through delegation, authorizes the command  
537 and parameters, on error return TPM\_AUTHFAIL
- 538 2. Determine the type of inputData (TPM\_DELEGATE\_TABLE\_ROW or  
539 TPM\_DELEGATE\_OWNER\_BLOB or TPM\_DELEGATE\_KEY\_BLOB) and map D1 to that  
540 structure
- 541 a. Mapping to TPM\_DELEGATE\_TABLE\_ROW requires taking inputData as a tableIndex  
542 and locating the appropriate row in the table
- 543 3. If D1 is a TPM\_DELEGATE\_OWNER\_BLOB or TPM\_DELEGATE\_KEY\_BLOB, validate the  
544 integrity of D1
- 545 a. Copy D1 -> integrityDigest to H2
- 546 b. Set D1 -> integrityDigest to NULL
- 547 c. Create H3 the HMAC of D1 using tpmProof as the secret
- 548 d. Compare H2 to H3 return TPM\_AUTHFAIL on mismatch
- 549 4. Locate (D1 -> pub -> familyID) in the TPM\_FAMILY\_TABLE and set familyRow to indicate  
550 row, return TPM\_BADINDEX if not found
- 551 5. Set FR to TPM\_FAMILY\_TABLE.famTableRow[familyRow]
- 552 6. If delegated, verify that family of the delegated Owner-auth is the same as D1:  
553 (authHandle -> familyID) == (D1 -> pub -> familyID); otherwise return error  
554 TPM\_DELEGATE\_FAMILY
- 555 7. If delegated, verify that the family of the delegated Owner-auth is enabled: if (authHandle  
556 -> familyID -> flags TPM\_FAMFLAG\_ENABLED) is FALSE, return TPM\_DISABLED\_CMD
- 557 8. Set D1 -> verificationCount to FR -> verificationCount

- 558 9. If D1 is a TPM\_DELEGATE\_OWNER\_BLOB or TPM\_DELEGATE\_KEY\_BLOB set the  
559 integrity of D1
- 560 a. Set D1 -> integrityDigest to NULL
- 561 b. Create H1 the HMAC of D1 using tpmProof as the secret
- 562 c. Set D1 -> integrityDigest to H1
- 563 10.If D1 is a blob recreate the blob and return it

**19.7 TPM\_Delegate\_VerifyDelegation****Start of informative comment:**

TPM\_VerifyDelegation interprets a delegate blob and returns success or failure, depending on whether the blob is currently valid. The delegate blob is NOT loaded into the TPM.

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal, TPM_Delegate_VerifyDelegation
4	4	2S	4	UINT32	delegationSize	The length of the delegated information blob
5	<>	3S	<>	BYTE[]	delegation	TPM_DELEGATE_KEY_BLOB or TPM_DELEGATE_OWNER_BLOB

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal, TPM_Delegate_VerifyDelegation

**Actions**

1. Determine the type of blob, If delegation -> tag is equal to TPM\_TAG\_DELGATE\_OWNER\_BLOB then
  - a. Map D1 a TPM\_DELEGATE\_OWNER\_BLOB to delegation
2. Else if delegation -> tag = TPM\_TAG\_DELG\_KEY\_BLOB
  - a. Map D1 a TPM\_DELEGATE\_KEY\_BLOB to delegation
3. Else return TPM\_BAD\_PARAMETER
4. Locate D1 -> familyID in the TPM\_FAMILY\_TABLE and set familyRow to indicate row, return TPM\_BADINDEX if not found
5. Set FR to TPM\_FAMILY\_TABLE.famTableRow[familyRow]
6. If FR -> flags TPM\_FAMFLAG\_ENABLED is FALSE, return TPM\_DISABLED\_CMD
7. Validate that D1 -> pub -> verificationCount matches FR -> verificationCount, on mismatch return TPM\_FAMILYCOUNT
8. Validate the integrity of D1
  - a. Copy D1 -> integrityDigest to H2

- 586        b. Set D1 -> integrityDigest to NULL
- 587        c. Create H3 the HMAC of D1 using tpmProof as the secret
- 588        d. Compare H2 to H3 return TPM\_AUTHFAIL on mismatch
- 589    9. Create S1 a TPM\_DELEGATE\_SENSITIVE area by decrypting D1 -> sensitiveArea using
- 590        TPM\_DELEGATE\_KEY
- 591    10. Validate S1 values
- 592        a. S1 -> tag is TPM\_TAG\_DELEGATE\_SENSITIVE
- 593        b. Return TPM\_BAD\_PARAMETER on error
- 594    11. Return TPM\_SUCCESS



## 20. Non-volatile Storage

**Start of informative comment:**

This section handles the allocation and use of the TPM non-volatile storage.

**End of informative comment.**

If nvIndex refers to the DIR, the TPM ignores actions containing access control checks that have no meaning for the DIR. The TPM only checks the owner authorization.

## 20.1 TPM\_NV\_DefineSpace

### Start of informative comment:

This establishes the space necessary for the indicated index. The definition will include the access requirements for writing and reading the area.

The space definition size does not include the area needed to manage the space.

Setting TPM\_PERMANENT\_FLAGS -> nvLocked TRUE when it is already TRUE is not an error.

### End of informative comment.

## Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Ordinal, TPM_ORD_NV_DefineSpace
4	<>	2S	<>	TPM_NV_DATA_PUBLIC	pubInfo	The public parameters of the NV area
5	20	3S	20	TPM_ENCAUTH	encAuth	The encrypted AuthData, only valid if the attributes require subsequent authorization
6	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for ownerAuth
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
7	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
9	20			TPM_AUTHDATA	ownerAuth	The authorization session digest HMAC key: ownerAuth

## Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	ordinal, TPM_ORD_NV_DefineSpace
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, fixed to FALSE
6	20			TPM_AUTHDATA	ownerAuth	The authorization session digest HMAC key: ownerAuth

## Actions

1. If pubInfo -> nvIndex == TPM\_NV\_INDEX\_LOCK and tag = TPM\_TAG\_RQU\_COMMAND

614       a. If pubInfo -> dataSize is not 0, the command MAY return TPM\_BADINDEX.  
615       b. Set TPM\_PERMANENT\_FLAGS -> nvLocked to TRUE  
616       c. Return TPM\_SUCCESS

617   2. If TPM\_PERMANENT\_FLAGS -> nvLocked is FALSE then all authorization checks except  
618       for the Max NV writes are ignored

619       a. Ignored checks include physical presence, authorization, 'D' bit check, index 0,  
620       bGlobalLock, no authorization with a TPM owner present, and bWriteSTClear

621   3. If pubInfo -> nvIndex has the D bit (bit 28) set to a 1 or pubInfo -> nvIndex == 0 then  
622       a. Return TPM\_BADINDEX  
623       b. The D bit specifies an index value that is set in manufacturing and can never be  
624       deleted or added to the TPM  
625       c. Index value of 0 is reserved and cannot be defined

626   4. If tag = TPM\_TAG\_RQU\_AUTH1\_COMMAND then  
627       a. The TPM MUST validate the command and parameters using the TPM Owner  
628       authentication and ownerAuth, on error return TPM\_AUTHFAIL  
629       b. authHandle session type MUST be OSAP  
630       c. If authHandle indicates XOR encryption for the AuthData secrets  
631           i. Create X1 the SHA-1 of the concatenation of (authHandle -> sharedSecret ||  
632           authLastNonceEven)  
633           ii. Create a1 by XOR X1 and encAuth  
634       d. Else  
635           i. Create a1 by decrypting encAuth using the algorithm indicated in the OSAP  
636           session  
637           ii. Key is from authHandle -> sharedSecret  
638           iii. IV is SHA-1 of (authLastNonceEven || nonceOdd)

639   5. else  
640       a. Validate the assertion of physical presence. Return TPM\_BAD\_PRESENCE on error.  
641       b. If TPM Owner is present then return TPM\_OWNER\_SET.  
642       c. If pubInfo -> dataSize is 0 then return TPM\_BAD\_DATASIZE. Setting the size to 0  
643       represents an attempt to delete the value without TPM Owner authentication.  
644       d. Validate max NV writes without an owner  
645           i. Set NV1 to TPM\_PERMANENT\_DATA -> noOwnerNVWrite  
646           ii. Increment NV1 by 1  
647           iii. If NV1 > TPM\_MAX\_NV\_WRITE\_NOOWNER return TPM\_MAXNVWRITES  
648           iv. Set TPM\_PERMANENT\_DATA -> noOwnerNVWrite to NV1  
649       e. Set A1 to encAuth. There is no nonce or authorization to create the encryption string,  
650       hence the AuthData value is passed in the clear

- 651 6. If pubInfo -> nvIndex points to a valid previously defined storage area then  
652 a. Map D1 a TPM\_NV\_DATA\_SENSITIVE to the storage area  
653 b. If D1 -> attributes specifies TPM\_NV\_PER\_GLOBALLOCK then  
654 i. If TPM\_STCLEAR\_FLAGS -> bGlobalLock is TRUE then return  
655 TPM\_AREA\_LOCKED  
656 c. If D1 -> attributes specifies TPM\_NV\_PER\_WRITE\_STCLEAR  
657 i. If D1 -> pubInfo -> bWriteSTClear is TRUE then return TPM\_AREA\_LOCKED  
658 d. Invalidate the data area currently pointed to by D1 and ensure that if the area is  
659 reallocated no residual information is left  
660 e. The TPM invalidates authorization sessions  
661 i. MUST invalidate all authorization sessions associated with D1  
662 ii. MAY invalidate any other authorization session  
663 f. If pubInfo -> dataSize is 0 then return TPM\_SUCCESS  
664 7. Parse pubInfo -> pcrInfoRead  
665 a. Validate pcrInfoRead structure on error return TPM\_INVALID\_STRUCTURE  
666 i. Validation includes proper PCR selections and locality selections  
667 8. Parse pubInfo -> pcrInfoWrite  
668 a. Validate pcrInfoWrite structure on error return TPM\_INVALID\_STRUCTURE  
669 i. Validation includes proper PCR selections and locality selections  
670 b. If pcrInfoWrite -> localityAtRelease disallows some localities  
671 i. Set writeLocalities to TRUE  
672 c. Else  
673 i. Set writeLocalities to FALSE  
674 9. Validate that the attributes are consistent  
675 a. The TPM SHALL ignore the bReadSTClear, bWriteSTClear and bWriteDefine  
676 attributes during the execution of this command  
677 b. If TPM\_NV\_PER\_OWNERWRITE is TRUE and TPM\_NV\_PER\_AUTHWRITE is TRUE  
678 return TPM\_AUTH\_CONFLICT  
679 c. If TPM\_NV\_PER\_OWNERREAD is TRUE and TPM\_NV\_PER\_AUTHREAD is TRUE  
680 return TPM\_AUTH\_CONFLICT  
681 d. If TPM\_NV\_PER\_OWNERWRITE and TPM\_NV\_PER\_AUTHWRITE and  
682 TPM\_NV\_PER\_WRITEDEFINE and TPM\_NV\_PER\_PPWRITE and writeLocalities are all  
683 FALSE  
684 i. Return TPM\_PER\_NOWRITE  
685 e. Validate nvIndex  
686 i. Make sure that the index is applicable for this TPM return TPM\_BADINDEX on  
687 error. A valid index is platform and context sensitive. That is attempting to

688           validate an index may be successful in one configuration and invalid in another  
689           configuration. The individual index values MUST indicate if there are any  
690           restrictions on the use of the index.

691       f. If dataSize is 0 return TPM\_BAD\_PARAM\_SIZE

692   10. Create D1 a TPM\_NV\_DATA\_SENSITIVE structure

693   11. Validate that sufficient NV is available to store the data

694       a. return TPM\_NOSPACE if pubInfo -> dataSize is not available in the TPM

695   12. Ensure that the TPM reserves the space for dataSize

696       a. Set all bytes in the newly defined area to 0xFF

697   13. Set D1 -> pubInfo to pubInfo

698   14. Set D1 -> authValue to A1

699   15. Set D1 -> pubInfo -> bReadSTClear = FALSE;

700   16. Set D1 -> pubInfo -> bWriteSTClear = FALSE;

701   17. Set D1 -> pubInfo -> bWriteDefine = FALSE;

702   18. Ignore continueAuthSession on input and set to FALSE on output

703   19. Return TPM\_SUCCESS

## 20.2 TPM\_NV\_WriteValue

### Start of informative comment:

This command writes the value to a defined area. The write can be TPM Owner authorized or unauthorized and protected by other attributes and will work when no TPM Owner is present.

### End of informative comment.

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Ordinal, TPM_ORD_NV_WriteValue
4	4	2S	4	TPM_NV_INDEX	nvIndex	The index of the area to set
5	4	3S	4	UINT32	offset	The offset into the NV Area
6	4	4S	4	UINT32	dataSize	The size of the data parameter
7	<>	5S	<>	BYTE	data	The data to set the area to
8	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for TPM Owner
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
9	20	3H1	20	TPM_NONCE	authNonceOdd	Nonce generated by caller
10	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
11	20			TPM_AUTHDATA	ownerAuth	The authorization session digest HMAC key: ownerAuth

### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	ordinal, TPM_ORD_NV_WriteValue
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	authNonceOdd	Nonce generated by caller
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	ownerAuth	The authorization session digest HMAC key: ownerAuth

### Actions

- If TPM\_PERMANENT\_FLAGS -> nvLocked is FALSE then all authorization checks except for the max NV writes are ignored
  - Ignored checks include physical presence, authorization, TPM\_NV\_PER\_OWNERWRITE, and PCR

717 2. If nvIndex = 0 then  
718 a. If dataSize is not 0, return TPM\_BADINDEX.  
719 b. Set TPM\_STCLEAR\_FLAGS -> bGlobalLock to TRUE  
720 c. Return TPM\_SUCCESS  
721 3. Locate and set D1 to the TPM\_NV\_DATA\_AREA that corresponds to nvIndex, return  
722 TPM\_BADINDEX on error  
723 a. If nvIndex = TPM\_NV\_INDEX\_DIR, set D1 to TPM\_PERMANENT\_DATA -> authDir[0]  
724 4. If D1 -> permission -> TPM\_NV\_PER\_AUTHWRITE is TRUE return  
725 TPM\_AUTH\_CONFLICT  
726 5. If tag = TPM\_TAG\_RQU\_AUTH1\_COMMAND then  
727 a. If D1 -> permission -> TPM\_NV\_PER\_OWNERWRITE is FALSE return  
728 TPM\_AUTH\_CONFLICT  
729 b. Validate command and parameters using ownerAuth HMAC with TPM Owner  
730 authentication as the secret, return TPM\_AUTHFAIL on error  
731 6. Else  
732 a. If D1 -> permission -> TPM\_NV\_PER\_OWNERWRITE is TRUE return  
733 TPM\_AUTH\_CONFLICT  
734 b. If no TPM Owner validate max NV writes without an owner  
735 i. Set NV1 to TPM\_PERMANENT\_DATA -> noOwnerNVWrite  
736 ii. Increment NV1 by 1  
737 iii. If NV1 > TPM\_MAX\_NV\_WRITE\_NOOWNER return TPM\_MAXNVWRITES  
738 iv. Set TPM\_PERMANENT\_DATA -> noOwnerNVWrite to NV1  
739 7. Check that D1 -> pcrInfoWrite -> localityAtRelease for TPM\_STANY\_DATA ->  
740 localityModifier is TRUE  
741 a. For example if TPM\_STANY\_DATA -> localityModifier was 2 then D1 -> pcrInfo ->  
742 localityAtRelease -> TPM\_LOC\_TWO would have to be TRUE  
743 b. On error return TPM\_BAD\_LOCALITY  
744 8. If D1 -> attributes specifies TPM\_NV\_PER\_PPWRITE then validate physical presence is  
745 asserted if not return TPM\_BAD\_PRESENCE  
746 9. If D1 -> attributes specifies TPM\_NV\_PER\_WRITEDEFINE  
747 a. If D1 -> bWriteDefine is TRUE return TPM\_AREA\_LOCKED  
748 10. If D1 -> attributes specifies TPM\_NV\_PER\_GLOBALLOCK  
749 a. If TPM\_STCLEAR\_DATA -> bGlobalLock is TRUE return TPM\_AREA\_LOCKED  
750 11. If D1 -> attributes specifies TPM\_NV\_PER\_WRITE\_STCLEAR  
751 a. If D1 -> bWriteSTClear is TRUE return TPM\_AREA\_LOCKED  
752 12. If D1 -> pcrInfoWrite -> pcrSelection specifies a selection of TPM\_STCLEAR\_DATA ->  
753 PCR[]

754       a. Create P1 a composite hash of the TPM\_STCLEAR\_DATA -> PCR[] specified by D1 ->  
755       pcrInfoWrite  
756       b. Compare P1 to D1 -> pcrInfoWrite -> digestAtRelease return TPM\_WRONGPCRVAL  
757       on mismatch  
758   13.If dataSize = 0 then  
759       a. Set D1 -> bWriteSTClear to TRUE  
760       b. Set D1 -> bWriteDefine to TRUE  
761   14.Else  
762       a. Set S1 to offset + dataSize  
763       b. If S1 > D1 -> dataSize return TPM\_NOSPACE  
764       c. If D1 -> attributes specifies TPM\_NV\_PER\_WRITEALL  
765           i. If dataSize != D1 -> dataSize return TPM\_NOT\_FULLWRITE  
766       d. Write the new value into the NV storage area  
767   15.Set D1 -> bReadSTClear to FALSE  
768   16.Return TPM\_SUCCESS



**20.3 TPM\_NV\_WriteValueAuth****Start of informative comment:**

This command writes to a previously defined area. The area must require authorization to write. Use this command when authorization other than the owner authorization is to be used. Otherwise, use TPM\_NV\_WriteValue.

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	Tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Ordinal, TPM_ORD_NV_WriteValueAuth
4	4	2S	4	TPM_NV_INDEX	nvIndex	The index of the area to set
5	4	3S	4	UINT32	offset	The offset into the chunk
6	4	4S	4	UINT32	dataSize	The size of the data area
7	◇	5S	◇	BYTE	data	The data to set the area to
8	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for NV element authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
9	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
10	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
11	20			TPM_AUTHDATA	authValue	HMAC key: NV element auth value

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	ordinal, TPM_ORD_NV_WriteValueAuth
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	NonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	authValue	HMAC key: NV element auth value

**Actions**

1. Locate and set D1 to the TPM\_NV\_DATA\_AREA that corresponds to nvIndex, return TPM\_BADINDEX on error
2. If D1 -> attributes does not specify TPM\_NV\_PER\_AUTHWRITE then return TPM\_AUTH\_CONFLICT

- 782 3. Validate authValue using D1 -> authValue, return TPM\_AUTHFAIL on error
- 783 4. Check that D1 -> pcrInfoWrite -> localityAtRelease for TPM\_STANY\_DATA ->
- 784 localityModifier is TRUE
- 785 a. For example if TPM\_STANY\_DATA -> localityModifier was 2 then D1 -> pcrInfo ->
- 786 localityAtRelease -> TPM\_LOC\_TWO would have to be TRUE
- 787 b. On error return TPM\_BAD\_LOCALITY
- 788 5. If D1 -> attributes specifies TPM\_NV\_PER\_PPWRITE then validate physical presence is
- 789 asserted if not return TPM\_BAD\_PRESENCE
- 790 6. If D1 -> pcrInfoWrite -> pcrSelection specifies a selection of PCR
- 791 a. Create P1 a composite hash of the TPM\_STCLEAR\_DATA -> PCR[] specified by D1 ->
- 792 pcrInfoWrite
- 793 b. Compare P1 to digestAtRelease return TPM\_WRONGPCRVAL on mismatch
- 794 7. If D1 -> attributes specifies TPM\_NV\_PER\_WRITEDEFINE
- 795 a. If D1 -> bWriteDefine is TRUE return TPM\_AREA\_LOCKED
- 796 8. If D1 -> attributes specifies TPM\_NV\_PER\_GLOBALLOCK
- 797 a. If TPM\_STCLEAR\_FLAGS -> bGlobalLock is TRUE return TPM\_AREA\_LOCKED
- 798 9. If D1 -> attributes specifies TPM\_NV\_PER\_WRITE\_STCLEAR
- 799 a. If D1 -> bWriteSTClear is TRUE return TPM\_AREA\_LOCKED
- 800 10.If dataSize = 0 then
- 801 a. Set D1 -> bWriteSTClear to TRUE
- 802 b. Set D1 -> bWriteDefine to TRUE
- 803 11.Else
- 804 a. Set S1 to offset + dataSize
- 805 b. If S1 > D1 -> dataSize return TPM\_NOSPACE
- 806 c. If D1 -> attributes specifies TPM\_NV\_PER\_WRITEALL
- 807 i. If dataSize != D1 -> dataSize return TPM\_NOT\_FULLWRITE
- 808 d. Write the new value into the NV storage area
- 809 12.Set D1 -> bReadSTClear to FALSE
- 810 13.Return TPM\_SUCCESS

**20.4 TPM\_NV\_ReadValue****Start of informative comment:**

Read a value from the NV store. This command uses optional owner authentication.

Action 1 indicates that if the NV are is not locked then reading of the NV area continues without ANY authorization. This is intentional and allows a platform manufacturer to set the NV areas, read them back, and then lock them all without having to install a TPM owner.

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Ordinal, TPM_ORD_NV_ReadValue
4	4	2S	4	TPM_NV_INDEX	nvIndex	The index of the area to set
5	4	3S	4	UINT32	offset	The offset into the area
6	4	4S	4	UINT32	dataSize	The size of the data area
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for TPM Owner authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	authNonceOdd	Nonce generated by caller
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
10	20			TPM_AUTHDATA	ownerAuth	HMAC key: ownerAuth

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S		TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	TPM_ORD_NV_ReadValue
4	4	3S	4	UINT32	dataSize	The size of the data area
5	◁	4S	◁	BYTE	data	The data to set the area to
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20			TPM_AUTHDATA	ownerAuth	HMAC key: ownerAuth

**Actions**

- 822 1. If TPM\_PERMANENT\_FLAGS -> nvLocked is FALSE then all authorization checks are  
823 ignored
- 824 2. Set D1 a TPM\_NV\_DATA\_AREA structure to the area pointed to by nvIndex, if not found  
825 return TPM\_BADINDEX
- 826 a. If nvIndex = TPM\_NV\_INDEX\_DIR, set D1 to TPM\_PERMANENT\_DATA -> authDir[0]
- 827 3. If tag = TPM\_TAG\_RQU\_AUTH1\_COMMAND then
- 828 a. If D1 -> TPM\_NV\_PER\_OWNERREAD is FALSE return TPM\_AUTH\_CONFLICT
- 829 b. Validate command and parameters using TPM Owners authentication on error return  
830 TPM\_AUTHFAIL
- 831 4. Else
- 832 a. If D1 -> TPM\_NV\_PER\_AUTHREAD is TRUE return TPM\_AUTH\_CONFLICT
- 833 b. If D1 -> TPM\_NV\_PER\_OWNERREAD is TRUE return TPM\_AUTH\_CONFLICT
- 834 5. Check that D1 -> pcrInfoRead -> localityAtRelease for TPM\_STANY\_DATA ->  
835 localityModifier is TRUE
- 836 a. For example if TPM\_STANY\_DATA -> localityModifier was 2 then D1 -> pcrInfo ->  
837 localityAtRelease -> TPM\_LOC\_TWO would have to be TRUE
- 838 b. On error return TPM\_BAD\_LOCALITY
- 839 6. If D1 -> attributes specifies TPM\_NV\_PER\_PPREAD then validate physical presence is  
840 asserted if not return TPM\_BAD\_PRESENCE
- 841 7. If D1 -> TPM\_NV\_PER\_READ\_STCLEAR then
- 842 a. If D1 -> bReadSTClear is TRUE return TPM\_DISABLED\_CMD
- 843 8. If D1 -> pcrInfoRead -> pcrSelection specifies a selection of PCR
- 844 a. Create P1 a composite hash of the TPM\_STCLEAR\_DATA -> PCR[] specified by D1 ->  
845 pcrInfoRead
- 846 b. Compare P1 to D1 -> pcrInfoRead -> digestAtRelease return TPM\_WRONGPCRVAL on  
847 mismatch
- 848 9. If dataSize is 0 then
- 849 a. Set D1 -> bReadSTClear to TRUE
- 850 b. Set data to NULL
- 851 10. Else
- 852 a. Set S1 to offset + dataSize
- 853 b. If S1 > D1 -> dataSize return TPM\_NOSPACE
- 854 c. Set data to area pointed to by offset
- 855 11. Return TPM\_SUCCESS

**20.5 TPM\_NV\_ReadValueAuth****Start of informative comment:**

This command requires that the read be authorized by a value set with the blob.

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Ordinal, TPM_ORD_NV_ReadValueAuth
4	4	2S	4	TPM_NV_INDEX	nvIndex	The index of the area to set
5	4	3S	4	UINT32	offset	The offset from the data area
6	4	5S	4	UINT32	dataSize	The size of the data area
7	4			TPM_AUTHHANDLE	authHandle	authThe auth handle for the NV element authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	authNonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	authContinueSession	The continue use flag for the authorization session handle
10	20			TPM_AUTHDATA	authHmac	HMAC key: nv element authorization

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	ordinal, TPM_ORD_NV_ReadValueAuth
4	4	3S	4	UINT32	dataSize	The size of the data area
5	<>	4S	<>	BYTE	data	The data
6	20	2H1	20	TPM_NONCE	authNonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	authLastNonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	authContinueSession	Continue use flag, TRUE if handle is still active
8	20			TPM_AUTHDATA	authHmacOut	HMAC key: nv element authorization

**Actions**

1. Locate and set D1 to the TPM\_NV\_DATA\_AREA that corresponds to nvIndex, on error return TPM\_BADINDEX
2. If D1 -> TPM\_NV\_PER\_AUTHREAD is FALSE return TPM\_AUTH\_CONFLICT
3. Validate authHmac using D1 -> authValue on error return TPM\_AUTHFAIL

- 367 4. If D1 -> attributes specifies TPM\_NV\_PER\_PPREAD then validate physical presence is  
368 asserted if not return TPM\_BAD\_PRESENCE
- 369 5. Check that D1 -> pcrInfoRead -> localityAtRelease for TPM\_STANY\_DATA ->  
370 localityModifier is TRUE
- 371 a. For example if TPM\_STANY\_DATA -> localityModifier was 2 then D1 -> pcrInfo ->  
372 localityAtRelease -> TPM\_LOC\_TWO would have to be TRUE
- 373 b. On error return TPM\_BAD\_LOCALITY
- 374 6. If D1 -> pcrInfoRead -> pcrSelection specifies a selection of PCR
- 375 a. Create P1 a composite hash of the TPM\_STCLEAR\_DATA -> PCR[] specified by D1 ->  
376 pcrInfoRead
- 377 b. Compare P1 to D1 -> pcrInfoRead -> digestAtRelease return TPM\_WRONGPCRVAL on  
378 mismatch
- 379 7. If D1 specifies TPM\_NV\_PER\_READ\_STCLEAR then
- 380 a. If D1 -> bReadSTClear is TRUE return TPM\_DISABLED\_CMD
- 381 8. If dataSize is 0 then
- 382 a. Set D1 -> bReadSTClear to TRUE
- 383 b. Set data to NULL
- 384 9. Else
- 385 a. Set S1 to offset + dataSize
- 386 b. If S1 > D1 -> dataSize return TPM\_NOSPACE
- 387 c. Set data to area pointed to by offset
- 388 10. Return TPM\_SUCCESS

## 21. Session Management

### Start of informative comment:

Three TPM\_RT\_CONTEXT session resources located in TPM\_STANY\_DATA work together to control session save and load: contextNonceSession, contextCount, and contextList[].

All three MUST be initialized at TPM\_Startup(ST\_ANY), which invalidates all saved sessions. They MAY be restored by TPM\_Startup(ST\_STATE), which would allow saved sessions to be loaded. The operation is reported as the TPM\_RT\_CONTEXT startup effect.

TPM\_SaveContext creates a contextBlob containing an encrypted contextNonceSession. The nonce is checked by TPM\_LoadContext. So initializing contextNonceSession invalidates all saved contexts. The nonce is large and protected, making a replay infeasible.

The contextBlob also contains a public but protected contextCount. The count increments for each saved contextBlob. The TPM also saves contextCount in contextList[]. The TPM validates contextBlob against the contextList[] during TPM\_LoadContext. Since the contextList[] is finite, it limits the number of valid saved sessions. Since the contextCount cannot be allowed to wrap, it limits the total number of saved sessions.

After a contextBlob is loaded, its contextCount entry is removed from contextList[]. This releases space in the context list for future entries. It also invalidates the contextBlob. So a saved contextBlob can be loaded only once.

TPM\_FlushSpecific can also specify a contextCount to be removed from the contextList[], allowing invalidation of an individual contextBlob. This is different from TPM\_FlushSpecific specifying a session handle, which invalidates a loaded session, not a saved contextBlob.

### End of informative comment.

### 21.1 TPM\_KeyControlOwner

#### Start of informative comment:

This command controls some attributes of keys that are stored within the TPM key cache.

OwnerEvict: If this bit is set to true, this key remains in the TPM through all TPM\_Startup events. The only way to evict this key is for the TPM Owner to execute this command again, setting the owner control bit to false and then executing TPM\_FlushSpecific.

The key handle does not reference an authorized entity and is not validated.

#### End of informative comment.

### Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes incl. paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_KeyControlOwner
4	4			TPM_KEY_HANDLE	keyHandle	The handle of a loaded key.

5	◁	2S	◁	TPM_PUBKEY	pubKey	The public key associated with the loaded key
6	4	3S	4	TPM_KEY_CONTROL	bitName	The name of the bit to be modified
7	1	4S	1	BOOL	bitValue	The value to set the bit to
8	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication
9		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
10	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
11	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
12	20		20	TPM_AUTHDATA	ownerAuth	HMAC authorization: key ownerAuth

## 921 Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal:TPM_ORD_KeyControlOwner
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM.
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	HMAC authorization: key ownerAuth

## 922 Descriptions

- 923 1. Set an internal bit within the key cache that controls some attribute of a loaded key.

## 924 Actions

- 925 1. Validate the AuthData using the owner authentication value, on error return  
926 TPM\_AUTHFAIL
- 927 2. Validate that keyHandle refers to a loaded key, return TPM\_INVALID\_KEYHANDLE on  
928 error.
- 929 3. Validate that pubKey matches the key held by the TPM pointed to by keyHandle, return  
930 TPM\_BAD\_PARAMETER on mismatch
- 931 a. This check added so that virtualization of the keyHandle does not result in attacks as  
932 the keyHandle is not associated with an authorization value
- 933 4. Validate that bitName is valid, return TPM\_BAD\_MODE on error.
- 934 5. If bitName == TPM\_KEY\_CONTROL\_OWNER\_EVICT
- 935 a. If bitValue == TRUE
- 936 i. Verify that after this operation at least two key slots will be present within the  
937 TPM that can store any type of key both of which do NOT have the OwnerEvict bit  
938 set, on error return TPM\_NOSPACE



939           ii. Verify that for this key handle, parentPCRStatus is FALSE and isVolatile is  
940           FALSE. Return TPM\_BAD\_PARAMETER on error.

941           iii. Set ownerEvict within the internal key storage structure to TRUE.

942       b. Else if bitValue == FALSE

943           i. Set ownerEvict within the internal key storage structure to FALSE.

944   6. Return TPM\_SUCCESS

## 21.2 TPM\_SaveContext

### Start of informative comment:

TPM\_SaveContext saves a loaded resource outside the TPM. After successful execution of the command, the TPM automatically releases the internal memory for sessions but leaves keys in place.

There is no assumption that a saved context blob is stored in a safe, protected area. Since the context blob can be loaded at any time, do not rely on TPM\_SaveContext to restrict access to an entity such as a key. If use of the entity should be restricted, means such as authorization secrets or PCR's should be used.

In general, TPM\_SaveContext can save a transport session. However, it cannot save an exclusive transport session, because any ordinal other than TPM\_ExecuteTransport terminates the exclusive transport session. This action prevents the exclusive transport session from being saved and reloaded while intervening commands are hidden from the transport log.

### End of informative comment.

## Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveContext
4	4			TPM_HANDLE	handle	Handle of the resource being saved.
5	4	2S	4	TPM_RESOURCE_TYPE	resourceType	The type of resource that is being saved
6	16	3S	16	BYTE[16]	label	Label for identification purposes

## Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveContext
4	4	3S	4	UINT32	contextSize	The actual size of the outgoing context blob
5	<>	4S	<>	TPM_CONTEXT_BLOB	contextBlob	The context blob

## Description

- The caller of the function uses the label field to add additional sequencing, anti-replay or other items to the blob. The information does not need to be confidential but needs to be part of the blob integrity.

966 **Actions**

- 967 1. Map V1 to TPM\_STANY\_DATA
- 968 2. Validate that handle points to resource that matches resourceType, return  
969 TPM\_INVALID\_RESOURCE on error
- 970 3. Validate that resourceType is a resource from the following list if not return  
971 TPM\_INVALID\_RESOURCE
- 972 a. TPM\_RT\_KEY
- 973 b. TPM\_RT\_AUTH
- 974 c. TPM\_RT\_TRANS
- 975 d. TPM\_RT\_DAA\_TPM
- 976 4. Locate the correct nonce
- 977 a. If resourceType is TPM\_RT\_KEY
- 978 i. If TPM\_STCLEAR\_DATA -> contextNonceKey is NULLS
- 979 (1) Set TPM\_STCLEAR\_DATA -> contextNonceKey to the next value from the TPM  
980 RNG
- 981 ii. Map N1 to TPM\_STCLEAR\_DATA -> contextNonceKey
- 982 iii. If the key has TPM\_KEY\_CONTROL\_OWNER\_EVICT set then return  
983 TPM\_OWNER\_CONTROL
- 984 b. Else
- 985 i. If V1 -> contextNonceSession is NULLS
- 986 (1) Set V1 -> contextNonceSession to the next value from the TPM RNG
- 987 ii. Map N1 to V1 -> contextNonceSession
- 988 5. Set K1 to TPM\_PERMANENT\_DATA -> contextKey
- 989 6. Create R1 by putting the sensitive part of the resource pointed to by handle into a  
990 structure. The structure is a TPM manufacturer option. The TPM MUST ensure that ALL  
991 sensitive information of the resource is included in R1.
- 992 7. Create C1 a TPM\_CONTEXT\_SENSITIVE structure
- 993 a. C1 forms the inner encrypted wrapper for the blob. All saved context blobs MUST  
994 include a TPM\_CONTEXT\_SENSITIVE structure and the TPM\_CONTEXT\_SENSITIVE  
995 structure MUST be encrypted.
- 996 b. Set C1 -> contextNonce to N1
- 997 c. Set C1 -> internalData to R1
- 998 8. Create B1 a TPM\_CONTEXT\_BLOB
- 999 a. Set B1 -> tag to TPM\_TAG\_CONTEXTBLOB
- 000 b. Set B1 -> resourceType to resourceType
- 001 c. Set B1 -> handle to handle
- 002 d. Set B1 -> integrityDigest to NULL

003 e. Set B1 -> label to label  
004 f. Set B1 -> additionalData to information determined by the TPM manufacturer. This  
005 data will help the TPM to reload and reset context. This area MUST NOT hold any data  
006 that is sensitive (symmetric IV are fine, prime factors of an RSA key are not).  
007 i. For OSAP sessions, and DSAP attached to keys, the hash of the entity MUST be  
008 included in additionalData  
009 g. Set B1 -> additionalSize to the size of additionalData  
010 h. Set B1 -> sensitiveSize to the size of C1  
011 i. Set B1 -> sensitiveData to C1  
012 9. If resourceType is TPM\_RT\_KEY  
013 a. Set B1 -> contextCount to 0  
014 10. Else  
015 a. If V1 -> contextCount >  $2^{32}-2$  then  
016 i. Return with TPM\_TOOMANYCONTEXTS  
017 b. Else  
018 i. Increment V1 -> contextCount by 1  
019 ii. Validate that the TPM can still manage the new count value  
020 (1) If the distance between the oldest saved context and the contextCount is too  
021 large return TPM\_CONTEXT\_GAP  
022 iii. Find contextIndex such that V1 -> contextList[contextIndex] equals 0. If not found  
023 exit with TPM\_NOCONTEXTSPACE  
024 iv. Set V1-> contextList[contextIndex] to V1 -> contextCount  
025 v. Set B1 -> contextCount to V1 -> contextCount  
026 c. The TPM MUST invalidate all information regarding the resource except for  
027 information needed for reloading  
028 11. Calculate B1 -> integrityDigest the HMAC of B1 using TPM\_PERMANENT\_DATA ->  
029 tpmProof as the secret  
030 12. Create E1 by encrypting C1 using K1 as the key  
031 a. Set B1 -> sensitiveSize to the size of E1  
032 b. Set B1 -> sensitiveData to E1  
033 13. Set contextSize to the size of B1  
034 14. Return B1 in contextBlob

## 21.3 TPM\_LoadContext

### Start of informative comment:

TPM\_LoadContext loads into the TPM a previously saved context. The command returns a handle.

### End of informative comment.

### Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadContext
4	4			TPM_HANDLE	entityHandle	The handle the TPM MUST use to locate the entity tied to the OSAP/DSAP session
5	1	2S	1	BOOL	keepHandle	Indication if the handle MUST be preserved
6	4	3S	4	UINT32	contextSize	The size of the following context blob.
7	<	4S	<	TPM_CONTEXT_BLOB	contextBlob	The context blob

### Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadContext
4	4			TPM_HANDLE	handle	The handle assigned to the resource after it has been successfully loaded.

### Actions

1. Map contextBlob to B1, a TPM\_CONTEXT\_BLOB structure
2. Map V1 to TPM\_STANY\_DATA
3. Create M1 by decrypting B1 -> sensitiveData using TPM\_PERMANENT\_DATA -> contextKey
4. Create C1 and R1 by splitting M1 into a TPM\_CONTEXT\_SENSITIVE structure and internal resource data
5. Check contextNonce
  - a. If B1 -> resourceType is NOT TPM\_RT\_KEY
    - i. If C1 -> contextNonce does not equal V1 -> contextNonceSession return TPM\_BADCONTEXT

053           ii. Validate that the resource pointed to by the context is loaded (i.e. for OSAP the  
054           key referenced is loaded and DSAP connected to the key) return  
055           TPM\_RESOURCEMISSING

056           (1) For OSAP sessions the TPM MUST validate that the incoming pubkey hash  
057           matches the key held by the TPM

058           (2) For OSAP and DSAP sessions referring to a key, verify that entityHandle  
059           identifies the key linked to this OSAP/DSAP session, if not return  
060           TPM\_BAD\_HANDLE.

061       b. Else

062           i. If C1 -> internalData -> parentPCRStatus is FALSE and C1 -> internalData ->  
063           isVolatile is FALSE

064           (1) Ignore C1 -> contextNonce

065           ii. else

066           (1) If C1 -> contextNonce does not equal TPM\_STCLEAR\_DATA ->  
067           contextNonceKey return TPM\_BADCONTEXT

068   6. Validate the structure

069       a. Set H1 to B1 -> integrityDigest

070       b. Set B1 -> integrityDigest to NULL

071       c. Copy M1 to B1 -> sensitiveData

072       d. Create H2 the HMAC of B1 using TPM\_PERMANENT\_DATA -> tpmProof as the HMAC  
073       key

074       e. If H2 does equal H1 return TPM\_BADCONTEXT

075   7. If keepHandle is TRUE

076       a. Set handle to B1 -> handle

077       b. If the TPM is unable to restore the handle the TPM MUST return TPM\_BAD\_HANDLE

078   8. Else

079       a. The TPM SHOULD attempt to restore the handle but if not possible it MAY set the  
080       handle to any valid for B1 -> resourceType

081   9. If B1 -> resourceType is NOT TPM\_RT\_KEY

082       a. Find contextIndex such that V1 -> contextList[contextIndex] equals B1 ->  
083       TPM\_CONTEXT\_BLOB -> contextCount

084       b. If not found then return TPM\_BADCONTEXT

085       c. Set V1 -> contextList[contextIndex] to 0

086   10. Process B1 to return the resource back into TPM use

## 22. Eviction

**Start of informative comment:**

The TPM has numerous resources held inside of the TPM that may need eviction. The need for eviction occurs when the number of resources in use by the TPM exceed the available space. For resources that are hard to reload (i.e. keys tied to PCR values) the outside entity should first perform a context save before evicting items.

In version 1.1 there were separate commands to evict separate resource types. This new command set uses the resource types defined for context saving and creates a generic command that will evict all resource types.

**End of informative comment.**

The TPM MUST NOT flush the EK or SRK using this command.

Version 1.2 deprecates the following commands:

? TPM\_Terminate\_Handle

? TPM\_EvictKey

? TPM\_Reset

## 22.1 TPM\_FlushSpecific

### Start of informative comment:

TPM\_FlushSpecific flushes from the TPM a specific handle.

### End of informative comment.

### Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_FlushSpecific
4	4			TPM_HANDLE	handle	The handle of the item to flush
5	4	2S	4	TPM_RESOURCE_TYPE	resourceType	The type of resource that is being flushed

### Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_FlushSpecific

### Description

TPM\_FlushSpecific releases the resources associated with the given handle.

### Actions

1. If resourceType is TPM\_RT\_CONTEXT
  - a. The handle for a context is not a handle but the "context count" value. The TPM uses the "context count" value to locate the proper contextList entry and sets R1 to the contextList entry
  - b. If R1 is not a valid saved context return TPM\_BAD\_PARAMETER
2. Else if resourceType is TPM\_RT\_KEY
  - a. Set R1 to the key pointed to by handle
  - b. Validate that R1 points at valid key
  - c. If R1 -> ownerEvict is TRUE return TPM\_KEY\_OWNER\_CONTROL
3. Else if resourceType is TPM\_RT\_HASH or TPM\_RT\_COUNTER or TPM\_RT\_DELEGATE
  - a. Return TPM\_INVALID\_RESOURCE
4. Else



- 123        a. Set R1 to the resource pointed to by handle
- 124        b. Validate that resource type and handle point to a valid allocated resource
- 125    5. Invalidate R1 and all internal resources allocated to R1
- 126        a. Resources include authorization sessions

## 23. Timing Ticks

### Start of informative comment:

The TPM timing ticks are always available for use. The association of timing ticks to actual time is a protocol that occurs outside of the TPM. See the design document for details.

The setting of the clock type variable is a one time operation that allows the TPM to be configured to the type of platform that is installed on.

The ability for the TPM to continue to increment the timer ticks across power cycles of the platform is a TPM and platform manufacturer decision.

### End of informative comment.

## 23.1 TPM\_GetTicks

### Start of informative comment:

This command returns the current tick count of the TPM.

### End of informative comment.

### Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Ordinal: TPM_ORD_GetTicks

### Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Ordinal: TPM_ORD_GetTicks
4	32	3S	32	TPM_CURRENT_TICKS	currentTime	The current time held in the TPM

### Descriptions

This command returns the current time held in the TPM. It is the responsibility of the external system to maintain any relation between this time and a UTC value or local real time value.

### Actions

1. Set T1 to the internal TPM\_CURRENT\_TICKS structure
2. Return T1 as currentTime.

**23.2 TPM\_TickStampBlob****Start of informative comment:**

This command applies a time stamp to the passed blob. The TPM makes no representation regarding the blob merely that the blob was present at the TPM at the time indicated.

**End of informative comment.****Incoming Parameters and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Ordinal, fixed value of TPM_ORD_TickStampBlob
4	4			TPM_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key that can perform digital signatures.
5	20	2s	20	TPM_NONCE	antiReplay	Anti replay value to added to signature
6	20	3s	20	TPM_DIGEST	digestToStamp	The digest to perform the tick stamp on
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
10	20			TPM_AUTHDATA	privAuth	The authorization session digest that authorizes the use of keyHandle. HMAC key: key.usageAuth

## 155 Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Ordinal, fixed value of TPM_ORD_TickStampBlob
4	32	3S	32	TPM_CURRENT_TICKS	currentTicks	The current time according to the TPM
5	4	4S	4	UINT32	sigSize	The length of the returned digital signature
6	◇	5S	◇	BYTE[]	sig	The resulting digital signature.
7	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
9	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: key.usageAuth

## 156 Description

157 The function performs a digital signature on the hash of digestToStamp and the current tick  
158 count.

159 It is the responsibility of the external system to maintain any relation between tick count  
160 and a UTC value or local real time value.

## 161 Actions

- 162 3. The TPM validates the AuthData to use the key pointed to by keyHandle.
- 163 4. Validate that keyHandle -> keyUsage is TPM\_KEY\_SIGNING, TPM\_KEY\_IDENTITY or  
164 TPM\_KEY\_LEGACY, if not return the error code TPM\_INVALID\_KEYUSAGE.
- 165 5. Return TPM\_INAPPROPRIATE\_SIG if the keyHandle -> sigScheme is not SHA-1
- 166 6. If TPM\_STCLEAR\_DATA -> currentTicks is not properly initialized  
167 a. Initialize the TPM\_STCLEAR\_DATA -> currentTicks
- 168 7. Create T1, a TPM\_CURRENT\_TICKS structure.
- 169 8. Create H1 a TPM\_SIGN\_INFO structure and set the structure defaults  
170 a. Set H1 -> fixed to "TSTP"  
171 b. Set H1 -> replay to antiReplay  
172 c. Create H2 the concatenation of digestToStamp || T1  
173 d. Set H1 -> dataLen to the length of H2  
174 e. Set H1 -> data to H2
- 175 9. The TPM computes the signature, sig, using the key referenced by keyHandle, using  
176 SHA-1 of H1 as the information to be signed

177 10. The TPM returns T1 as currentTicks parameter

## 24. Transport Sessions

### 24.1 TPM\_EstablishTransport

#### Start of informative comment:

This establishes the transport session. Depending on the attributes specified for the session this may establish shared secrets, encryption keys, and session logs. The session will be in use for by the TPM\_ExecuteTransport command.

The only restriction on what can happen inside of a transport session is that there is no “nesting” of sessions. It is permissible to perform operations that delete internal state and make the TPM inoperable.

#### End of informative comment.

#### Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_EstablishTransport
4	4			TPM_KEY_HANDLE	encHandle	The handle to the key that encrypted the blob
5	<>	2S	<>	TPM_TRANSPORT_PUBLIC	transPublic	The public information describing the transport session
6	4	3S	4	UINT32	secretSize	The size of the secret Area
7	<>	4S	<>	BYTE[]	secret	The encrypted secret area
8	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
9	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
10	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
11	20			TPM_AUTHDATA	keyAuth	Authorization. HMAC key: encKey.usageAuth

190 **Outgoing Parameters and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_EstablishTransport
4	4			TPM_TRANSHANDLE	transHandle	The handle for the transport session
5	4	3S	4	TPM_MODIFIER_INDICATOR	locality	The locality that called this command
6	32	4S	32	TPM_CURRENT_TICKS	currentTicks	The current tick count
7	20	5S	20	TPM_NONCE	transNonceEven	The even nonce in use for subsequent execute transport
8	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
10	20			TPM_AUTHDATA	resAuth	Authorization. HMAC key: key.usageAuth

191 **Description**

192 This command establishes the transport sessions shared secret. The encryption of the  
 193 shared secret uses the public key of the key loaded in encKey.

194 **Actions**

- 195 1. If encHandle is TPM\_KH\_TRANSPORT then
  - 196 a. If tag is NOT TPM\_TAG\_RQU\_COMMAND return TPM\_BADTAG
  - 197 b. If transPublic -> transAttributes specifies TPM\_TRANSPORT\_ENCRYPT return  
 198 TPM\_BAD\_SCHEME
  - 199 c. If secretSize is not 20 return TPM\_BAD\_PARAM\_SIZE
  - 200 d. Set A1 to secret
- 201 2. Else
  - 202 a. encHandle -> keyUsage MUST be TPM\_KEY\_STORAGE or TPM\_KEY\_LEGACY return  
 203 TPM\_INVALID\_KEYUSAGE on error
  - 204 b. If encHandle -> authDataUsage does not equal TPM\_AUTH\_NEVER and tag is NOT  
 205 TPM\_TAG\_RQU\_AUTH1\_COMMAND return TPM\_AUTHFAIL
  - 206 c. Using encHandle -> usageAuth validate the AuthData to use the key and the  
 207 parameters to the command
  - 208 d. Create K1 a TPM\_TRANSPORT\_AUTH structure by decrypting secret using the key  
 209 pointed to by encHandle
  - 210 e. Validate K1 for tag
  - 211 f. Set A1 to K1 -> authData
- 212 3. If transPublic -> transAttributes has TPM\_TRANSPORT\_ENCRYPT

- 213 a. If TPM\_PERMANENT\_FLAGS -> FIPS is true and transPublic -> algId is equal to  
214 TPM\_ALG\_MGF1 return TPM\_INAPPROPRIATE\_ENC
- 215 b. Check if the transPublic -> algId is supported, if not return  
216 TPM\_BAD\_KEY\_PROPERTY
- 217 c. If transPublic -> algId is TPM\_ALG\_3DES or TPM\_ALG\_AESXXX, check that  
218 transPublic -> encScheme is supported, if not return TPM\_INAPPROPRIATE\_ENC
- 219 d. Perform any initializations necessary for the algorithm
- 220 4. Generate transNonceEven from the TPM RNG
- 221 5. Create T1 a TPM\_TRANSPORT\_INTERNAL structure
- 222 a. Ensure that the TPM has sufficient internal space to allocate the transport session,  
223 return TPM\_RESOURCES on error
- 224 b. Assign a T1 -> transHandle value. This value is assigned by the TPM
- 225 c. Set T1 -> transDigest to NULL
- 226 d. Set T1 -> transPublic to transPublic
- 227 e. Set T1-> transNonceEven to transNonceEven
- 228 f. Set T1 -> authData to A1
- 229 6. If TPM\_STANY\_DATA -> currentTicks is not properly initialized
- 230 a. Initialize the TPM\_STANY\_DATA -> currentTicks
- 231 7. Set currentTicks to TPM\_STANY\_DATA -> currentTicks
- 232 8. If T1 -> transPublic -> transAttributes has TPM\_TRANSPORT\_LOG set then
- 233 a. Create L1 a TPM\_TRANSPORT\_LOG\_IN structure
- 234 i. Set L1 -> parameters to SHA-1 (ordinal || transPublic || secretSize || secret)
- 235 ii. Set L1 -> pubKeyHash to NULL
- 236 iii. Set T1 -> transDigest to SHA-1 (T1 -> transDigest || L1)
- 237 b. Create L2 a TPM\_TRANSPORT\_LOG\_OUT structure
- 238 i. Set L2 -> parameters to SHA-1 (returnCode || ordinal || locality || currentTicks  
239 || transNonceEven)
- 240 ii. Set L2 -> locality to the locality of this command
- 241 iii. Set L2 -> currentTicks to currentTicks, this MUST be the same value that is  
242 returned in the currentTicks parameter
- 243 iv. Set T1 -> transDigest to SHA-1 (T1 -> transDigest || L2)
- 244 9. If T1 -> transPublic -> transAttributes has TPM\_TRANSPORT\_EXCLUSIVE then set  
245 TPM\_STANY\_FLAGS -> transportExclusive to TRUE
- 246 a. Execution of any command other than TPM\_ExecuteTransport or  
247 TPM\_ReleaseTransportSigned targeting this transport session will cause the abnormal  
248 invalidation of this transport session transHandle



- 249        b. The TPM gives no indication, other than invalidation of transHandle, that the session  
250        is terminated
- 251    10.Return T1 -> transHandle as transHandle

## 24.2 TPM\_ExecuteTransport

### Start of informative comment:

Delivers a wrapped TPM command to the TPM where the TPM unwraps the command and then executes the command.

TPM\_ExecuteTransport uses the same rolling nonce paradigm as other authorized TPM commands. The even nonces start in TPM\_EstablishTransport and change on each invocation of TPM\_ExecuteTransport.

The only restriction on what can happen inside of a transport session is that there is no “nesting” of sessions. It is permissible to perform operations that delete internal state and make the TPM inoperable.

Because, in general, key handles are not logged, a digest of the corresponding public key is logged. In cases where the key handle is logged (e.g. TPM\_OwnerReadInternalPub), the public key is also logged.

### End of informative comment.

### Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ExecuteTransport
4	4	2S	4	UINT32	wrappedCmdSize	Size of the wrapped command
5	<>	3S	<>	BYTE[]	wrappedCmd	The wrapped command
6	4			TPM_TRANSHANDLE	transHandle	The transport session handle
		2H1	20	TPM_NONCE	transLastNonceEven	Even nonce previously generated by TPM
7	20	3H1	20	TPM_NONCE	transNonceOdd	Nonce generated by caller
8	1	4H1	1	BOOL	continueTransSession	The continue use flag for the authorization session handle
9	20			TPM_AUTHDATA	transAuth	HMAC for transHandle key: transHandle -> authData

268 **Outgoing Parameters and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the ExecuteTransport command. This does not reflect the status of wrapped command.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ExecuteTransport
4	8	3S	8	UINT64	currentTicks	The current ticks when the command was executed
5	4	4S	4	TPM_MODIFIER_INDICATOR	locality	The locality that called this command
6	4	5S	4	UINT32	wrappedRspSize	Size of the wrapped response
7	◊	6S	◊	BYTE[]	wrappedRsp	The wrapped response
8	20	2H1	20	TPM_NONCE	transNonceEven	Even nonce newly generated by TPM
		3H1	20	TPM_NONCE	transNonceOdd	Nonce generated by caller
9	1	4H1	1	BOOL	continueTransSession	The continue use flag for the session
10	20			TPM_AUTHDATA	transAuth	HMAC for transHandle key: transHandle -> authData

269 **Description**

- 270 1. This command executes a TPM command using the transport session.
- 271 2. Prior to execution of the wrapped command (action 11 below) failure of the transport
- 272 session MUST have no effect on the resources referenced by the wrapped command. The
- 273 exception is when the TPM goes into failure mode and return FAILED\_SELFTEST for all
- 274 subsequent commands.
- 275 3. After execution of the wrapped command, failure of the transport session MUST have an
- 276 effect on the wrapped command resources. The reason for this is that the transport
- 277 session will be returning an error code and not reporting any session nonces. The entire
- 278 wrapped command response is lost so nonces, handles and such are lost to the caller.
- 279 4. Execution of the wrapped command (action 11) SHOULD have no effect on the transport
- 280 session.
- 281 a. The wrapped command SHALL use no resources of the transport session, this
- 282 includes authorization sessions
- 283 b. If the wrapped command execution returns an error (action 11 below) then the
- 284 sessions for TPM\_ExecuteTransport still operate properly.
- 285 c. The exception to this is when the wrapped command causes the TPM to go into
- 286 failure mode and return TPM\_FAILSELFTEST for all subsequent commands
- 287 5. Field layout
- 288 a. Command representation
- 289 b. \*\*\*\*\*
- 290 c. TAGet | LENet | ORDet | wrappedCmdSize | wrappedCmd | AUTHet
- 291 d. \*\*\*\*\*

e. wrappedCmd looks like the following

f. \*\*\*\*\*

g. TAGw | LENw | ORDw | HANDLESw(o) | DATAw | AUTH1w (o) | AUTH2w (o)

h. \*\*\*\*\*

i. | LEN1 |

j. | E1 | (encrypted)

k. | C1 | (decrypted)

l. Response representation

m. \*\*\*\*\*

n. TAGet | LENet | RCet | wrappedRspSize | wrappedRsp | AUTHet

o. \*\*\*\*\*

p. wrappedRsp looks like the following

q. \*\*\*\*\*

r. TAGw | LENw | RCw | HANDLESw(o) | DATAw | AUTH1w (o) | AUTH2w (o)

s. \*\*\*\*\*

t. | LEN2 |

u. | ←----- C2 -----→ |

v. | S2 | (decrypted)

w. | E2 | (encrypted)

x. The only parameter that is possibly encrypted is DATAw

## 6. Additional DATAw comments

a. For TPM\_FlushSpecific and TPM\_SaveContext

i. The DATAw part of these commands does not include the handle.

(1) It is understood that encrypting the resourceType prevents a determination of the handle type.

ii. If the resourceType is TPM\_RT\_KEY, then the public key SHOULD be logged.

b. For TPM\_DAA\_Join and TPM\_DAA\_Sign

i. The DATAw part of these commands does not include the handle

c. For TPM\_LoadKey2

i. The outgoing handle is not part of the outgoing DATAw and is not encrypted or logged by the outgoing transport.

d. For TPM\_LoadKey

i. The outgoing handle is part of the outgoing DATAw and is encrypted.

e. For TPM\_LoadContext

- i. The outgoing handle is not part of the outgoing DATAw and is not encrypted or logged by the outgoing transport.

- (1) It is understood that encrypting the contextBlob prevents a determination of the handle type.

- 7. TPM\_ExecuteTransport returns an implementation defined result when the wrapped command would cause termination of the transport session. Implementation defined possibilities include but are not limited to: the wrapped command may execute, completely, partially, or not at all, the transport session may or not be terminated, continueTransSession may not be processed or returned correctly, and an error may or may not be returned. The wrapped commands include:

- a. TPM\_FlushSpecific, TPM\_SaveContext targeting the transport session
- b. TPM\_OwnerClear, TPM\_ForceClear, TPM\_RevokeTrust

## Actions

1. Using transHandle locate the TPM\_TRANSPORT\_INTERNAL structure T1
2. Parse wrappedCmd
  - a. Set TAGw, LENw, and ORDw to the parameters from wrappedCmd
  - b. Set E1 to DATAw
    - i. This pointer is ordinal dependent and requires the execute transport command to parse wrappedCmd
  - c. Set LEN1 to the length of DATAw
    - i. DATAw always ends at the start of AUTH1w if AUTH1w is present
3. If LEN1 is less than 0, or if ORDw is unknown, unimplemented, or cannot be determined
  - a. Return TPM\_BAD\_PARAMETER
4. If T1 -> transPublic -> transAttributes has TPM\_TRANSPORT\_ENCRYPT set then
  - a. If T1 -> transPublic -> algId is TPM\_ALG\_MGF1
    - i. Using the MGF1 function, create string G1 of length LEN1. The inputs to the MGF1 are transLastNonceEven, transNonceOdd, "in", and T1 -> authData. These four values concatenated together form the Z value that is the seed for the MGF1.
    - ii. Create C1 by performing an XOR of G1 and wrappedCmd starting at E1.
  - b. If the encryption algorithm requires an IV calculate the IV values
    - i. Using the MGF1 function, create string IV1 with a length set by the block size of the encryption algorithm. The inputs to the MGF1 are transLastNonceEven, transNonceOdd, and "in". These three values concatenated together form the Z value that is the seed for the MGF1. Note that any terminating characters within the string "in" are ignored, so a total of 42 bytes are hashed.
    - ii. Blocksize for TPM\_ALG\_DES is 8
    - iii. Blocksize for TPM\_ALG\_AESxxx is 16
    - iv. The symmetric key is taken from the first bytes of T1 -> authData.

- 364 v. Decrypt DATAw and replace the DATAw area of E1 creating C1
- 365 c. TPM\_OSAP, TPM\_OIAP have no parameters encrypted
- 366 d. TPM\_DSAP has special rules for parameter encryption
- 367 5. Else
- 368 a. Set C1 to the DATAw area E1 of wrappedCmd
- 369 6. Create H1 the SHA-1 of (ORDw || C1).
- 370 a. C1 MUST point at the decrypted DATAw area of E1
- 371 b. The TPM MAY use this calculation for both execute transport authorization,
- 372 authorization of the wrapped command and transport log creation
- 373 7. Validate the incoming transport session authorization
- 374 a. Set inParamDigest to SHA-1 (ORDet || wrappedCmdSize || H1)
- 375 b. Calculate the HMAC of (inParamDigest || transLastNonceEven || transNonceOdd ||
- 376 continueTransSession) using T1 -> authData as the HMAC key
- 377 c. Validate transAuth, on errors return TPM\_AUTHFAIL
- 378 8. If TPM\_ExecuteTransport requires auditing
- 379 a. Create TPM\_AUDIT\_EVENT\_IN using H1 as the input parameter digest and update
- 380 auditDigest
- 381 b. On any error return TPM\_AUDITFAIL\_UNSUCCESSFUL
- 382 9. If ORDw is from the list of following commands return TPM\_NO\_WRAP\_TRANSPORT
- 383 a. TPM\_EstablishTransport
- 384 b. TPM\_ExecuteTransport
- 385 c. TPM\_ReleaseTransportSigned
- 386 10. If T1 -> transPublic -> transAttributes has TPM\_TRANSPORT\_LOG set then
- 387 a. Create L2 a TPM\_TRANSPORT\_LOG\_IN structure
- 388 b. Set L2 -> parameters to H1
- 389 c. If ORDw is a command with no key handles
- 390 i. Set L2 -> pubKeyHash to NULL
- 391 d. If ORDw is a command with one key handle
- 392 i. Create K2 the hash of the TPM\_STORE\_PUBKEY structure of the key pointed to
- 393 by the key handle.
- 394 ii. Set L2 -> pubKeyHash to SHA-1 (K2)
- 395 e. If ORDw is a command with two key handles
- 396 i. Create K2 the hash of the TPM\_STORE\_PUBKEY structure of the key pointed to
- 397 by the first key handle.
- 398 ii. Create K3 the hash of the TPM\_STORE\_PUBKEY structure of the key pointed to
- 399 by the second key handle.

```

400         iii. Set L2 -> pubKeyHash to SHA-1 (K2 || K3)
401     f. Set T1 -> transDigest to the SHA-1 (T1 -> transDigest || L2)
402     g. If ORDw is a command with key handles, and the key is not loaded, return
403        TPM_INVALID_KEYHANDLE.
404 11. Send the wrapped command to the normal TPM command parser, the output is C2 and
405     the return code is RCw
406     a. If ORDw is a command that is audited then the TPM MUST perform the input and
407        output audit of the command as part of this action.
408     b. The TPM MAY use H1 as the data value in the authorization and audit calculations
409        during the execution of C1
410 12. Set CT1 to TPM_STANY_DATA -> currentTicks -> currentTicks and return CT1 in the
411     currentTicks output parameter
412 13. Calculate S2 the pointer to the DATAw area of C2
413     a. Calculate LEN2 the length of S2 according to the same rules that calculated LEN1
414 14. Create H2 the SHA-1 of (RCw || ORDw || S2)
415     a. The TPM MAY use this calculation for execute transport authorization and transport
416        log out creation
417 15. Calculate the outgoing transport session authorization
418     a. Create the new transNonceEven for the output of the command
419     b. Set outParamDigest to SHA-1 (RCet || ORDet || TPM_STANY_DATA -> currentTicks
420        -> currentTicks || locality || wrappedRspSize || H2)
421     c. Calculate transAuth, the HMAC of (outParamDigest || transNonceEven ||
422        transNonceOdd || continueTransSession) using T1 -> authData as the HMAC key
423 16. If T1 -> transPublic -> transAttributes has TPM_TRANSPORT_LOG set then
424     a. Create L3 a TPM_TRANSPORT_LOG_OUT structure
425     b. Set L3 -> parameters to H2
426     c. Set L3 -> currentTicks to TPM_STANY_DATA -> currentTicks
427     d. Set L3 -> locality to TPM_STANY_DATA -> localityModifier
428     e. Set T1 -> transDigest to the SHA-1 (T1 -> transDigest || L3)
429 17. If T1 -> transPublic -> transAttributes has TPM_TRANSPORT_ENCRYPT set then
430     a. If T1 -> transPublic -> AlgId is TPM_ALG_MGF1
431         i. Using the MGF1 function, create string G2 of length LEN2. The inputs to the
432            MGF1 are transNonceEven, transNonceOdd, "out", and T1 -> authData. These
433            four values concatenated together form the Z value that is the seed for the MGF1.
434         ii. Create E2 by performing an XOR of G2 and C2 starting at S2.
435     b. Else

```

436           i. Create IV2 using the same algorithm as IV1 with the input values  
437           transNonceEven, transNonceOdd, and "out". Note that any terminating  
438           characters within the string "out" are ignored, so a total of 43 bytes are hashed.

439           ii. Create E2 by encrypting C2 starting at S2 using IV2

440   18.Else

441       a. Set E2 to the DATAw area S2 of wrappedRsp

442   19.If continueTransSession is FALSE

443       a. Invalidate all session data related to transHandle

444   20.If TPM\_ExecuteTranport requires auditing

445       a. Create TPM\_AUDIT\_EVENT\_OUT using H2 for the parameters and update the  
446       auditDigest

447       b. On any errors return TPM\_AUDITFAIL\_SUCCESSFUL or  
448       TPM\_AUDITFAIL\_UNSUCCESSFUL depending on RCw

449   21.Return C2 but with S2 replaced by E2 in the wrappedRsp parameter



## 24.3 TPM\_ReleaseTransportSigned

**Start of informative comment:**

This command completes the transport session. If logging for this session is turned on, then this command returns a hash of all operations performed during the session along with a digital signature of the hash.

This command serves no purpose if logging is turned off, and results in an error if attempted.

This command uses two authorization sessions, the key that will sign the log and the authorization from the session. Having the session authorization proves that the requestor that is signing the log is the owner of the session. If this restriction is not put in then an attacker can close the log and sign using their own key.

The hash of the session log includes the information associated with the input phase of execution of the TPM\_ReleaseTransportSigned command. It cannot include the output phase information.

**End of informative comment.**

### Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReleaseTransportSigned
4	4			TPM_KEY_HANDLE	key Handle	Handle of a loaded key that will perform the signing
5	20	2S	20	TPM_NONCE	antiReplay	Value provided by caller for anti-replay protection
6	4			TPM_AUTHHANDLE	authHandle	The authorization session to use key
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
7	20	3H1	20	TPM_NONCE	authNonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
9	20			TPM_AUTHDATA	keyAuth	The authorization session digest that authorizes the use of key. HMAC key: key -> usageAuth
10	4			TPM_TRANSHANDLE	transHandle	The transport session handle
		2H2	20	TPM_NONCE	transLastNonceEven	Even nonce in use by execute Transport
11	20	3H2	20	TPM_NONCE	transNonceOdd	Nonce supplied by caller for transport session
12	1	4H2	1	BOOL	continueTrans Session	The continue use flag for the authorization session handle
13	20			TPM_AUTHDATA	transAuth	HMAC for transport session key: transHandle -> authData

## 466 Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReleaseTransportSigned
4	4	3S	4	TPM_MODIFIER_INDICATOR	locality	The locality that called this command
5	32	4S	32	TPM_CURRENT_TICKS	currentTicks	The current ticks when the command executed
6	4	5S	4	UINT32	signSize	The size of the signature area
7	<>	6S	<>	BYTE[]	signature	The signature of the digest
8	20	2H1	20	TPM_NONCE	authNonceEven	Even nonce newly generated by TPM
		3H1	20	TPM_NONCE	authNonceOdd	Nonce generated by caller
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the session
10	20			TPM_AUTHDATA	keyAuth	HMAC: key -> usageAuth
11	20	2H2	20	TPM_NONCE	transNonceEven	Even nonce newly generated by TPM
		3H2	20	TPM_NONCE	transNonceOdd	Nonce generated by caller
12	1	4H2	1	BOOL	continueTransSession	The continue use flag for the session
13	20			TPM_AUTHDATA	transAuth	HMAC: transHandle -> authData

## 467 Description

468 This command releases a transport session and signs the transport log

## 469 Actions

- 470 1. Using transHandle locate the TPM\_TRANSPORT\_INTERNAL structure T1
- 471 2. Return TPM\_INAPPROPRIATE\_SIG if the key -> sigScheme is not SHA-1
- 472 3. Using key -> authData validate the command and parameters, on error return
- 473 TPM\_AUTHFAIL
- 474 4. Using transHandle -> authData validate the command and parameters, on error return
- 475 TPM\_AUTH2FAIL
- 476 5. If T1 -> transAttributes has TPM\_TRANSPORT\_LOG set then
  - 477 a. Create A1 a TPM\_TRANSPORT\_LOG\_OUT structure
  - 478 b. Set A1 -> parameters to the SHA-1 (ordinal || antiReplay)
  - 479 c. Set A1 -> currentTicks to TPM\_STANY\_DATA -> currentTicks
  - 480 d. Set A1 -> locality to the locality modifier for this command
  - 481 e. Set T1 -> transDigest to SHA-1 (T1 -> transDigest || A1)
- 482 6. Else
- 483 a. Return TPM\_BAD\_MODE

- 484 7. Create H1 a TPM\_SIGN\_INFO structure and set the structure defaults
- 485     a. Set H1 -> fixed to "TRAN"
- 486     b. Set H1 -> replay to antiReplay
- 487     c. Set H1 -> data to T1 -> transDigest
- 488     d. Sign SHA-1 hash of H1 using the key pointed to by key
- 489 8. Invalidate all session data related to T1
- 490 9. Set continueTransSession to FALSE
- 491 10. Return TPM\_SUCCESS

## 25. Monotonic Counter

### 25.1 TPM\_CreateCounter

#### Start of informative comment:

This command creates the counter but does not select the counter. Counter creation assigns an AuthData value to the counter and sets the counters original start value. The original start value is the current internal base value plus one. Setting the new counter to the internal base avoids attacks on the system that are attempting to use old counter values.

#### End of informative comment.

### Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes incl. paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateCounter
4	20	2S	20	TPM_ENCAUTH	encAuth	The encrypted auth data for the new counter
5	4	3s	4	BYTE	label	Label to associate with counter
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	Ignored
10	20		20	TPM_AUTHDATA	ownerAuth	Authorization ownerAuth.

502 **Outgoing Parameters and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateCounter
4	4	3s	4	TPM_COUNT_ID	countID	The handle for the counter
5	10	4S	10	TPM_COUNTER_VALUE	counterValue	The starting counter value
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Fixed value of FALSE
8	20		20	TPM_AUTHDATA	resAuth	Authorization. HMAC key: ownerAuth.

503 **Description**

504 This command creates a new monotonic counter. The TPM MUST support a minimum of 4  
505 concurrent counters.

506 **Actions**

507 The TPM SHALL do the following:

- 508 1. Using the authHandle field, validate the owner's AuthData to execute the command and  
509 all of the incoming parameters. The authorization session MUST be OSAP or DSAP
- 510 2. Ignore continueAuthSession on input and set continueAuthSession to FALSE on output
- 511 3. If authHandle indicates XOR encryption for the AuthData secrets
  - 512 a. Create X1 the SHA-1 of the concatenation of (authHandle -> sharedSecret ||  
513 authLastNonceEven)
  - 514 b. Create a1 by XOR X1 and encAuth
- 515 4. Else
  - 516 a. Create a1 by decrypting encAuth using the algorithm indicated in the OSAP session
  - 517 b. Key is from authHandle -> sharedSecret
  - 518 c. IV is SHA-1 of (authLastNonceEven || nonceOdd)
- 519 5. Validate that there is sufficient internal space in the TPM to create a new counter. If  
520 there is insufficient space, the command returns an error.
  - 521 a. The TPM MUST provide storage for a1, TPM\_COUNTER\_VALUE, countID, and any  
522 other internal data the TPM needs to associate with the counter
- 523 6. Increment the max counter value
- 524 7. Set the counter to the max counter value
- 525 8. Set the counter label to label

526 9. Create a countID

**25.2 TPM\_IncrementCounter****Start of informative comment:**

This authorized command increments the indicated counter by one. Once a counter has been incremented then all subsequent increments must be for the same handle until a successful TPM\_Startup(ST\_CLEAR) is executed.

The order for checking validation of the command parameters when no counter is active, keeps an attacker from creating a denial-of-service attack.

**End of informative comment.****Incoming Parameters and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_IncrementCounter
4	4	2s	4	TPM_COUNT_ID	countID	The handle of a valid counter
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for counter authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
8	20			TPM_AUTHDATA	counterAuth	The authorization session digest that authorizes the use of countID. HMAC key: countID -> authData

**Outgoing Parameters and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_IncrementCounter
5	10	3S	10	TPM_COUNTER_VALUE	count	The counter value
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: countID -> authData

**Description**

This function increments the counter by 1.

The TPM MAY implement increment throttling to avoid burn problems

540 **Actions**

- 541 1. If TPM\_STCLEAR\_DATA -> countID is NULL
- 542 a. Validate that countID is a valid counter, return TPM\_BAD\_COUNTER on mismatch
- 543 b. Validate the command parameters using counterAuth
- 544 c. Set TPM\_STCLEAR\_DATA -> countID to countID
- 545 2. else
- 546 a. If TPM\_STCLEAR\_DATA -> countID does not equal countID
- 547 i. Return TPM\_BAD\_COUNTER
- 548 b. Validate the command parameters using counterAuth
- 549 3. Increments the counter by 1
- 550 4. Return new count value in count



**25.3 TPM\_ReadCounter****Start of informative comment:**

Reading the counter provides the caller with the current number in the sequence.

**End of informative comment.****Incoming Parameters and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes incl. paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReadCounter
4	4	2S	4	TPM_COUNT_ID	countID	ID value of the counter

**Outgoing Parameters and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReadCounter
4	10	3S	4	TPM_COUNTER_VALUE	count	The counter value

**Description**

This returns the current value for the counter indicated. The counter MAY be any valid counter.

**Actions**

1. Validate that countID points to a valid counter. Return TPM\_BAD\_COUNTER on error.
2. Return count

## 25.4 TPM\_ReleaseCounter

### Start of informative comment:

This command releases a counter such that no reads or increments of the indicated counter will succeed.

### End of informative comment.

### Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReleaseCounter
4	4	2s	4	TPM_COUNT_ID	countID	ID value of the counter
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for countID authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce associated with countID
7	1	4H1	1	BOOL	continueAuthSession	Ignored
8	20			TPM_AUTHDATA	counterAuth	The authorization session digest that authorizes the use of countID. HMAC key: countID -> authData

### Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReleaseCounter
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: countID -> authData

### Actions

The TPM uses countID to locate a valid counter.

1. Authenticate the command and the parameters using the AuthData pointed to by countID. Return TPM\_AUTHFAIL on error
2. The TPM invalidates all internal information regarding the counter. This includes releasing countID such that any subsequent attempts to use countID will fail.
3. The TPM invalidates sessions

- 577       a. MUST invalidate all OSAP sessions associated with the counter
- 578       b. MAY invalidate any other session
- 579   4. If TPM\_STCLEAR\_DATA -> countID equals countID,
- 580       a. Set TPM\_STCLEAR\_DATA -> countID to an illegal value (not the NULL value)

## 25.5 TPM\_ReleaseCounterOwner

### Start of informative comment:

This command releases a counter such that no reads or increments of the indicated counter will succeed.

### End of informative comment.

### Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReleaseCounterOwner
4	4	2s	4	TPM_COUNT_ID	countID	ID value of the counter
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
8	20			TPM_AUTHDATA	ownerAuth	The authorization session digest that authorizes the inputs. HMAC key: ownerAuth

### Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReleaseCounterOwner
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth

### Description

This invalidates all information regarding a counter.

### Actions

1. Validate that ownerAuth properly authorizes the command and parameters
2. The TPM uses countID to locate a valid counter. Return TPM\_BAD\_COUNTER if not found.

- 594 3. The TPM invalidates all internal information regarding the counter. This includes  
595 releasing countID such that any subsequent attempts to use countID will fail.
- 596 4. The TPM invalidates sessions
- 597 a. MUST invalidate all OSAP sessions associated with the counter
- 598 b. MAY invalidate any other session
- 599 5. If TPM\_STCLEAR\_DATA -> countID equals countID,
- 600 a. Set TPM\_STCLEAR\_DATA -> countID to an illegal value (not the NULL value)

## 26. DAA commands

### 26.1 TPM\_DAA\_Join

**Start of informative comment:**

TPM\_DAA\_Join is the process that establishes the DAA parameters in the TPM for a specific DAA issuing authority.

**End of informative comment.**

#### Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes incl. paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DAA_Join.
4	4			TPM_HANDLE	handle	Session handle
5	1	2S	1	BYTE	stage	Processing stage of join
6	4	3S	4	UINT32	inputSize0	Size of inputData0 for this stage of JOIN
7	◇	4S	◇	BYTE[]	inputData0	Data to be used by this capability
8	4	5S	4	UINT32	inputSize1	Size of inputData1 for this stage of JOIN
9	◇	6S	◇	BYTE[]	inputData1	Data to be used by this capability
10	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication
		2 H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
11	20	3 H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
12	1	4 H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
13	20		20	TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner. HMAC key: ownerAuth.

608 **Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes incl. paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DAA_Join.
4	4	3S	4	UINT32	outputSize	Size of outputData
5	<>	4S	<>	BYTE[]	outputData	Data produced by this capability
6	20	2 H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3 H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4 H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20		20	TPM_AUTHDATA	resAuth	Authorization HMAC key: ownerAuth.

## Description

This table summaries the input, output and saved data that is associated with each stage of processing.

Stage	Input Data0	Input Data1	Operation	Output Data	Scratchpad
0	DAA_count (used as # repetitions of stage 1)	NULL	initialise	Session Handle	NULL
1	n0	signatureValue	rekeying	NULL	n0
2	DAA_issuerSettings	signatureValue	issuer settings	NULL	NULL
3	DAA_count	NULL	DAA_join_uo, DAA_join_u1	NULL	NULL
4	DAA_generic_R0	DAA_generic_n	$P1 = R0^{f0} \bmod n$	NULL	P1
5	DAA_generic_R1	DAA_generic_n	$P2 = P1 * (R1^{f1}) \bmod n$	NULL	P2
6	DAA_generic_S0	DAA_generic_n	$P3 = P2 * (S0^{u0}) \bmod n$	NULL	P3
7	DAA_generic_S1	DAA_generic_n	$U = P3 * (S1^{u1}) \bmod n$	U	NULL
8	NE	NULL	U2	U2	NULL
9	DAA_generic_R0	DAA_generic_n	$P1 = R0^{f0} \bmod n$	NULL	P1
10	DAA_generic_R1	DAA_generic_n	$P2 = P1 * (R1^{f1}) \bmod n$	NULL	P2
11	DAA_generic_S0	DAA_generic_n	$P3 = P2 * (S0^{r2}) \bmod n$	NULL	P3
12	DAA_generic_S1	DAA_generic_n	$P4 = P3 * (S1^{r3}) \bmod n$	P4	NULL
13	DAA_generic_gamma	w	$w1 = w^q \bmod \text{gamma}$	NULL	w
14	DAA_generic_gamma	NULL	$E = w^f \bmod \text{gamma}$	E	w
15	DAA_generic_gamma	NULL	$r = r0 + (2^{\text{power0}})^{r1} \bmod q$ $E1 = w^r \bmod \text{gamma}$	E1	NULL
16	c1	NULL	$c = \text{hash}(c1 \parallel \text{NT})$	nt	NULL
17	NULL	NULL	$s0 = r0 + c^{f0}$	s0	NULL
18	NULL	NULL	$s1 = r1 + c^{f1}$	s1	NULL
19	NULL	NULL	$s2 = r2 + c^{u0}$ $\bmod 2^{\text{power1}}$	s2	NULL
20	NULL	NULL	$s12 = r2 + c^{u0}$ $\gg \text{power1}$	c	s12
21	NULL	NULL	$s3 = r3 + c^{u1} + s12$	s3	NULL
22	u2	NULL	$v0 = u2 + u0 \bmod 2^{\text{power1}}$ $v10 = u2 + u0 \gg \text{power1}$	enc(v0)	v10
23	u3	NULL	$V1 = u3 + u1 + v10$	enc(v1)	NULL
24	NULL	NULL	enc(DAA_tpmSpecific)	enc(DAA_tpmSpecific)	NULL



**613 Actions**

614 A Trusted Platform Module that receives a valid TPM\_DAA\_Join command SHALL:

- 615 1. Use ownerAuth to verify that the Owner authorized all TPM\_DAA\_Join input parameters.
- 616 2. Any error return results in the TPM invalidating all resources associated with the join
- 617 3. Constant values of 0 or 1 are 1 byte integers, stages affected are
  - 618 a. 4(j), 5(j), 14(f), 17(e)
- 619 4. Representation of the strings “r0” to “r4” are 2-byte ASCII encodings, stages affected are
  - 620 a. 9(i), 10(h), 11(h), 12(h), 15(f), 15(g), 17(d), 18(d), 19(d), 20(d), 21(d)

**621 Stages**

622 0. If stage==0

- 623 a. Determine that sufficient resources are available to perform a TPM\_DAA\_Join.
  - 624 i. The TPM MUST support sufficient resources to perform one (1) TPM\_DAA\_Join/  
625 TPM\_DAA\_Sign. The TPM MAY support additional TPM\_DAA\_Join/  
626 TPM\_DAA\_Sign sessions.
  - 627 ii. The TPM may share internal resources between the DAA operations and other  
628 variable resource requirements:
  - 629 iii. If there are insufficient resources within the stored key pool (and one or more  
630 keys need to be removed to permit the DAA operation to execute) return  
631 TPM\_NOSPACE
  - 632 iv. If there are insufficient resources within the stored session pool (and one or  
633 more authorization or transport sessions need to be removed to permit the  
634 DAA operation to execute), return TPM\_RESOURCES.
- 635 b. Set all fields in DAA\_issuerSettings = NULL
- 636 c. set all fields in DAA\_tpmSpecific = NULL
- 637 d. set all fields in DAA\_session = NULL
- 638 e. Set all fields in DAA\_JoinSession = NULL
- 639 f. Verify that sizeof(inputData0) == sizeof(DAA\_tpmSpecific -> DAA\_count) and return  
640 error TPM\_DAA\_INPUT\_DATA0 on mismatch
- 641 g. Verify that inputData0 > 0, and return error TPM\_DAA\_INPUT\_DATA0 on mismatch
- 642 h. Set DAA\_tpmSpecific -> DAA\_count = inputData0
- 643 i. set DAA\_session -> DAA\_digestContext = SHA-1(DAA\_tpmSpecific ||  
644 DAA\_joinSession))
- 645 j. set DAA\_session -> DAA\_stage = 1
- 646 k. Assign session handle for TPM\_DAA\_Join
- 647 l. set outputData = new session handle
- 648 m. return TPM\_SUCCESS

```

649 1. If stage==1
650     a. Verify that DAA_session ->DAA_stage==1. Return TPM_DAA_STAGE and flush handle
651        on mismatch
652     b. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
653        DAA_joinSession) and return TPM_DAA_TPM_SETTINGS on mismatch
654     c. Verify that sizeof(inputData0) == DAA_SIZE_issuerModulus and return error
655        TPM_DAA_INPUT_DATA0 on mismatch
656     d. If DAA_session -> DAA_scratch == NULL:
657         i. Set DAA_session -> DAA_scratch = inputData0
658         ii. set DAA_joinSession -> DAA_digest_n0 = SHA1(DAA_session -> DAA_scratch)
659         iii. set DAA_tpmSpecific -> DAA_rekey = SHA1(TPM_DAA_TPM_SEED ||
660             DAA_joinSession -> DAA_digest_n0)
661     e. Else (If DAA_session -> DAA_scratch != NULL):
662         i. Set signedData = inputData0
663         ii. Verify that sizeof(inputData1) == DAA_SIZE_issuerModulus and return error
664            TPM_DAA_INPUT_DATA1 on mismatch
665         iii. Set signatureValue = inputData1
666         iv. Use the RSA key == [DAA_session -> DAA_scratch] to verify that signatureValue is
667            a signature on signedData, and return error TPM_DAA_ISSUER_VALIDITY on
668            mismatch
669         v. Set DAA_session -> DAA_scratch = signedData
670     f. Decrement DAA_tpmSpecific -> DAA_count by 1 (unity)
671     g. If DAA_tpmSpecific -> DAA_count ==0:
672     h. increment DAA_Session -> DAA_Stage by 1
673     i. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific ||
674        DAA_joinSession)
675     j. set outputData = NULL
676     k. return TPM_SUCCESS
677 2. If stage==2
678     a. Verify that DAA_session ->DAA_stage==2. Return TPM_DAA_STAGE and flush handle
679        on mismatch
680     b. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
681        DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
682     c. Verify that sizeof(inputData0) == sizeof(TPM_DAA_ISSUER) and return error
683        TPM_DAA_INPUT_DATA0 on mismatch
684     d. Set DAA_issuerSettings = inputData0. Verify that all fields in DAA_issuerSettings are
685        present and return error TPM_DAA_INPUT_DATA0 if not.

```

```

686     e. Verify that sizeof(inputData1) == DAA_SIZE_issuerModulus and return error
687     TPM_DAA_INPUT_DATA1 on mismatch
688     f. Set signatureValue = inputData1
689     g. Set signedData = (DAA_joinSession -> DAA_digest_n0 || DAA_issuerSettings)
690     h. Use the RSA key [DAA_session -> DAA_scratch] to verify that signatureValue is a
691     signature on signedData, and return error TPM_DAA_ISSUER_VALIDITY on mismatch
692     i. Set DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings)
693     j. set   DAA_session   ->   DAA_digestContext   =   SHA-1(DAA_tpmSpecific   ||
694     DAA_joinSession)
695     k. Set DAA_session -> DAA_scratch = NULL
696     l. increment DAA_session -> DAA_stage by 1
697     m. return TPM_SUCCESS
698 3. If stage==3
699     a. Verify that DAA_session ->DAA_stage==3. Return TPM_DAA_STAGE and flush handle
700     on mismatch
701     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
702     return error TPM_DAA_ISSUER_SETTINGS on mismatch
703     c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
704     DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
705     d. Verify that sizeof(inputData0) == sizeof(DAA_tpmSpecific -> DAA_count) and return
706     error TPM_DAA_INPUT_DATA0 on mismatch
707     e. Set DAA_tpmSpecific -> DAA_count = inputData0
708     f. obtain random data from the RNG and store it as DAA_joinSession -> DAA_join_u0
709     g. obtain random data from the RNG and store it as DAA_joinSession -> DAA_join_u1
710     h. set outputData = NULL
711     i. increment DAA_session -> DAA_stage by 1
712     j. set   DAA_session   ->   DAA_digestContext   =   SHA-1(DAA_tpmSpecific   ||
713     DAA_joinSession)
714     k. return TPM_SUCCESS
715 4. If stage==4,
716     a. Verify that DAA_session ->DAA_stage==4. Return TPM_DAA_STAGE and flush handle
717     on mismatch
718     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
719     return error TPM_DAA_ISSUER_SETTINGS on mismatch
720     c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
721     DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
722     d. Set DAA_generic_R0 = inputData0

```

```

723     e. Verify that SHA-1(DAA_generic_R0) == DAA_issuerSettings -> DAA_digest_R0 and
724     return error TPM_DAA_INPUT_DATA0 on mismatch
725     f. Set DAA_generic_n = inputData1
726     g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and
727     return error TPM_DAA_INPUT_DATA1 on mismatch
728     h. Set X = DAA_generic_R0
729     i. Set n = DAA_generic_n
730     j. Set f = SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0
731     ) || SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1) mod
732     DAA_issuerSettings -> DAA_generic_q
733     k. Set f0 = f mod 2^DAA_power0 (erase all but the lowest DAA_power0 bits of f)
734     l. Set DAA_session -> DAA_scratch = (X^f0) mod n
735     m. set outputData = NULL
736     n. increment DAA_session -> DAA_stage by 1
737     o. return TPM_SUCCESS
738 5. If stage==5
739     a. Verify that DAA_session -> DAA_stage==5. Return TPM_DAA_STAGE and flush handle
740     on mismatch
741     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
742     return error TPM_DAA_ISSUER_SETTINGS on mismatch
743     c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
744     DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
745     d. Set DAA_generic_R1 = inputData0
746     e. Verify that SHA-1(DAA_generic_R1) == DAA_issuerSettings -> DAA_digest_R1 and
747     return error TPM_DAA_INPUT_DATA0 on mismatch
748     f. Set DAA_generic_n = inputData1
749     g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and
750     return error TPM_DAA_INPUT_DATA1 on mismatch
751     h. Set X = DAA_generic_R1
752     i. Set n = DAA_generic_n
753     j. Set f = SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0
754     ) || SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1) mod
755     DAA_issuerSettings -> DAA_generic_q.
756     k. Shift f right by DAA_power0 bits (discard the lowest DAA_power0 bits) and label the
757     result f1
758     l. Set Z = DAA_session -> DAA_scratch
759     m. Set DAA_session -> DAA_scratch = Z*(X^f1) mod n
760     n. set outputData = NULL

```

```

761     o. increment DAA_session -> DAA_stage by 1
762     p. return TPM_SUCCESS
763 6. If stage==6
764     a. Verify that DAA_session ->DAA_stage==6. Return TPM_DAA_STAGE and flush handle
765     on mismatch
766     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
767     return error TPM_DAA_ISSUER_SETTINGS on mismatch
768     c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
769     DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
770     d. Set DAA_generic_S0 = inputData0
771     e. Verify that SHA-1(DAA_generic_S0) == DAA_issuerSettings -> DAA_digest_S0 and
772     return error TPM_DAA_INPUT_DATA0 on mismatch
773     f. Set DAA_generic_n = inputData1
774     g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and
775     return error TPM_DAA_INPUT_DATA1 on mismatch
776     h. Set X = DAA_generic_S0
777     i. Set n = DAA_generic_n
778     j. Set Z = DAA_session -> DAA_scratch
779     k. Set Y = DAA_joinSession -> DAA_join_u0
780     l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n
781     m. set outputData = NULL
782     n. increment DAA_session -> DAA_stage by 1
783     o. return TPM_SUCCESS
784 7. If stage==7
785     a. Verify that DAA_session ->DAA_stage==7. Return TPM_DAA_STAGE and flush handle
786     on mismatch
787     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
788     return error TPM_DAA_ISSUER_SETTINGS on mismatch
789     c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
790     DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
791     d. Set DAA_generic_S1 = inputData0
792     e. Verify that SHA-1(DAA_generic_S1) == DAA_issuerSettings -> DAA_digest_S1 and
793     return error TPM_DAA_INPUT_DATA0 on mismatch
794     f. Set DAA_generic_n = inputData1
795     g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and
796     return error TPM_DAA_INPUT_DATA1 on mismatch
797     h. Set X = DAA_generic_S1

```

```

798     i. Set n = DAA_generic_n
799     j. Set Y = DAA_joinSession -> DAA_join_u1
800     k. Set Z = DAA_session -> DAA_scratch
801     l. Set DAA_session -> DAA_scratch =  $Z \cdot (X^Y) \bmod n$ 
802     m. Set DAA_session -> DAA_digest to the SHA-1 (DAA_session -> DAA_scratch ||
803     DAA_tpmSpecific -> DAA_count || DAA_joinSession -> DAA_digest_n0)
804     n. set outputData = DAA_session -> DAA_scratch
805     o. set DAA_session -> DAA_scratch = NULL
806     p. increment DAA_session -> DAA_stage by 1
807     q. return TPM_SUCCESS
808 8. If stage==8
809     a. Verify that DAA_session ->DAA_stage==8. Return TPM_DAA_STAGE and flush handle
810     on mismatch
811     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
812     return error TPM_DAA_ISSUER_SETTINGS on mismatch
813     c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
814     DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
815     d. Verify inputSize0 == DAA_SIZE_NE and return error TPM_DAA_INPUT_DATA0 on
816     mismatch
817     e. Set NE = decrypt(inputData0, privEK)
818     f. set outputData = SHA-1(DAA_session -> DAA_digest || NE)
819     g. set DAA_session -> DAA_digest = NULL
820     h. increment DAA_session -> DAA_stage by 1
821     i. return TPM_SUCCESS
822 9. If stage==9
823     a. Verify that DAA_session ->DAA_stage==9. Return TPM_DAA_STAGE and flush handle
824     on mismatch
825     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
826     return error TPM_DAA_ISSUER_SETTINGS on mismatch
827     c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
828     DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
829     d. Set DAA_generic_R0 = inputData0
830     e. Verify that SHA-1(DAA_generic_R0) == DAA_issuerSettings -> DAA_digest_R0 and
831     return error TPM_DAA_INPUT_DATA0 on mismatch
832     f. Set DAA_generic_n = inputData1
833     g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and
834     return error TPM_DAA_INPUT_DATA1 on mismatch

```

```

835     h. obtain random data from the RNG and store it as DAA_session -> DAA_contextSeed
836     i. obtain DAA_SIZE_r0 bits from MGF1("r0", DAA_session -> DAA_contextSeed), and
837        label them Y
838     j. Set X = DAA_generic_R0
839     k. Set n = DAA_generic_n
840     l. Set DAA_session -> DAA_scratch = (X^Y) mod n
841     m. set outputData = NULL
842     n. increment DAA_session -> DAA_stage by 1
843     o. return TPM_SUCCESS
844 10.If stage==10
845     a. Verify that DAA_session ->DAA_stage==10. Return TPM_DAA_STAGE and flush
846        handle on mismatch h
847     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
848        return error TPM_DAA_ISSUER_SETTINGS on mismatch
849     c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
850        DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
851     d. Set DAA_generic_R1 = inputData0
852     e. Verify that SHA-1(DAA_generic_R1) == DAA_issuerSettings -> DAA_digest_R1 and
853        return error TPM_DAA_INPUT_DATA0 on mismatch
854     f. Set DAA_generic_n = inputData1
855     g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and
856        return error TPM_DAA_INPUT_DATA1on mismatch
857     h. obtain DAA_SIZE_r1 bits from MGF1("r1", DAA_session -> DAA_contextSeed), and
858        label them Y
859     i. Set X = DAA_generic_R1
860     j. Set n = DAA_generic_n
861     k. Set Z = DAA_session -> DAA_scratch
862     l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n
863     m. set outputData = NULL
864     n. increment DAA_session -> DAA_stage by 1
865     o. return TPM_SUCCESS
866 11.If stage==11
867     a. Verify that DAA_session ->DAA_stage==11. Return TPM_DAA_STAGE and flush
868        handle on mismatch
869     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
870        return error TPM_DAA_ISSUER_SETTINGS on mismatch

```

871 c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific ||  
872 DAA\_joinSession) and return error TPM\_DAA\_TPM\_SETTINGS on mismatch  
873 d. Set DAA\_generic\_S0 = inputData0  
874 e. Verify that SHA-1(DAA\_generic\_S0) == DAA\_issuerSettings -> DAA\_digest\_S0 and  
875 return error TPM\_DAA\_INPUT\_DATA0 on mismatch  
876 f. Set DAA\_generic\_n = inputData1  
877 g. Verify that SHA-1(DAA\_generic\_n) == DAA\_issuerSettings -> DAA\_digest\_n and  
878 return error TPM\_DAA\_INPUT\_DATA1 on mismatch  
879 h. obtain DAA\_SIZE\_r2 bits from MGF1("r2", DAA\_session -> DAA\_contextSeed), and  
880 label them Y  
881 i. Set X = DAA\_generic\_S0  
882 j. Set n = DAA\_generic\_n  
883 k. Set Z = DAA\_session -> DAA\_scratch  
884 l. Set DAA\_session -> DAA\_scratch =  $Z \cdot (X^Y) \bmod n$   
885 m. set outputData = NULL  
886 n. increment DAA\_session -> DAA\_stage by 1  
887 o. return TPM\_SUCCESS  
888 12.If stage==12  
889 a. Verify that DAA\_session ->DAA\_stage==12. Return TPM\_DAA\_STAGE and flush  
890 handle on mismatch  
891 b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings ) and  
892 return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch  
893 c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific ||  
894 DAA\_joinSession) and return error TPM\_DAA\_TPM\_SETTINGS on mismatch  
895 d. Set DAA\_generic\_S1 = inputData0  
896 e. Verify that SHA-1(DAA\_generic\_S1) == DAA\_issuerSettings -> DAA\_digest\_S1 and  
897 return error TPM\_DAA\_INPUT\_DATA0 on mismatch  
898 f. Set DAA\_generic\_n = inputData1  
899 g. Verify that SHA-1(DAA\_generic\_n) == DAA\_issuerSettings -> DAA\_digest\_n and  
900 return error TPM\_DAA\_INPUT\_DATA1 on mismatch  
901 h. obtain DAA\_SIZE\_r3 bits from MGF1("r3", DAA\_session -> DAA\_contextSeed), and  
902 label them Y  
903 i. Set X = DAA\_generic\_S1  
904 j. Set n = DAA\_generic\_n  
905 k. Set Z = DAA\_session -> DAA\_scratch  
906 l. Set DAA\_session -> DAA\_scratch =  $Z \cdot (X^Y) \bmod n$   
907 m. set outputData = DAA\_session -> DAA\_scratch



```

908     n. Set DAA_session -> DAA_scratch = NULL
909     o. increment DAA_session -> DAA_stage by 1
910     p. return TPM_SUCCESS
911 13.If stage==13
912     a. Verify that DAA_session->DAA_stage==13. Return TPM_DAA_STAGE and flush
913        handle on mismatch
914     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
915        return error TPM_DAA_ISSUER_SETTINGS on mismatch
916     c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
917        DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
918     d. Set DAA_generic_gamma = inputData0
919     e. Verify that SHA-1(DAA_generic_gamma) == DAA_issuerSettings ->
920        DAA_digest_gamma and return error TPM_DAA_INPUT_DATA0 on mismatch
921     f. Verify that inputSize1 == DAA_SIZE_w and return error TPM_DAA_INPUT_DATA1 on
922        mismatch
923     g. Set w = inputData1
924     h. Set w1 = w^( DAA_issuerSettings -> DAA_generic_q) mod (DAA_generic_gamma)
925     i. If w1 != 1 (unity), return error TPM_DAA_WRONG_W
926     j. Set DAA_session -> DAA_scratch = w
927     k. set outputData = NULL
928     l. increment DAA_session -> DAA_stage by 1
929     m. return TPM_SUCCESS.
930 14.If stage==14
931     a. Verify that DAA_session ->DAA_stage==14. Return TPM_DAA_STAGE and flush
932        handle on mismatch
933     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings ) and
934        return error TPM_DAA_ISSUER_SETTINGS on mismatch
935     c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
936        DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
937     d. Set DAA_generic_gamma = inputData0
938     e. Verify that SHA-1(DAA_generic_gamma) == DAA_issuerSettings ->
939        DAA_digest_gamma and return error TPM_DAA_INPUT_DATA0 on mismatch
940     f. Set f = SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0
941        ) || SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1 ) mod
942        DAA_issuerSettings -> DAA_generic_q.
943     g. Set E = ((DAA_session -> DAA_scratch)^f) mod (DAA_generic_gamma).
944     h. Set outputData = E
945     i. increment DAA_session -> DAA_stage by 1

```

```
946     j. return TPM_SUCCESS.
947 15.If stage==15
948     a. Verify that DAA_session ->DAA_stage==15. Return TPM_DAA_STAGE and flush
949        handle on mismatch
950     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
951        return error TPM_DAA_ISSUER_SETTINGS on mismatch
952     c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
953        DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
954     d. Set DAA_generic_gamma = inputData0
955     e. Verify that SHA-1(DAA_generic_gamma) == DAA_issuerSettings ->
956        DAA_digest_gamma and return error TPM_DAA_INPUT_DATA0 on mismatch
957     f. obtain DAA_SIZE_r0 bits from MGF1("r0", DAA_session -> DAA_contextSeed), and
958        label them r0
959     g. obtain DAA_SIZE_r1 bits from MGF1("r1", DAA_session -> DAA_contextSeed), and
960        label them r1
961     h. set  $r = r0 + 2^{DAA\_power0} * r1 \bmod (DAA\_issuerSettings \rightarrow DAA\_generic\_q)$ .
962     i. set  $E1 = ((DAA\_session \rightarrow DAA\_scratch)^r) \bmod (DAA\_generic\_gamma)$ .
963     j. Set DAA_session -> DAA_scratch = NULL
964     k. Set outputData = E1
965     l. increment DAA_session -> DAA_stage by 1
966     m. return TPM_SUCCESS.
967 16.If stage==16
968     a. Verify that DAA_session ->DAA_stage==16. Return TPM_DAA_STAGE and flush
969        handle on mismatch
970     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
971        return error TPM_DAA_ISSUER_SETTINGS on mismatch
972     c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
973        DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
974     d. Verify that inputSize0 == sizeof(TPM_DIGEST) and return error
975        TPM_DAA_INPUT_DATA0 on mismatch
976     e. Set DAA_session -> DAA_digest = inputData0
977     f. obtain DAA_SIZE_NT bits from the RNG and label them NT
978     g. Set DAA_session -> DAA_digest to the SHA-1 ( DAA_session -> DAA_digest || NT )
979     h. Set outputData = NT
980     i. increment DAA_session -> DAA_stage by 1
981     j. return TPM_SUCCESS.
982 17.If stage==17
```

```

983     a. Verify that DAA_session ->DAA_stage==17. Return TPM_DAA_STAGE and flush
984     handle on mismatch
985     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
986     return error TPM_DAA_ISSUER_SETTINGS on mismatch
987     c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
988     DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
989     d. obtain DAA_SIZE_r0 bits from MGF1("r0", DAA_session -> DAA_contextSeed), and
990     label them r0
991     e. Set f = SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0
992     ) || SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1) mod
993     DAA_issuerSettings -> DAA_generic_q.
994     f. Set f0 = f mod 2^DAA_power0 (erase all but the lowest DAA_power0 bits of f)
995     g. Set s0 = r0 + (DAA_session -> DAA_digest) * f0 in Z
996     h. set outputData = s0
997     i. increment DAA_session -> DAA_stage by 1
998     j. return TPM_SUCCESS
999 18.If stage==18
000     a. Verify that DAA_session ->DAA_stage==18. Return TPM_DAA_STAGE and flush
001     handle on mismatch
002     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
003     return error TPM_DAA_ISSUER_SETTINGS on mismatch
004     c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
005     DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
006     d. obtain DAA_SIZE_r1 bits from MGF1("r1", DAA_session -> DAA_contextSeed), and
007     label them r1
008     e. Set f = SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0
009     ) || SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1) mod
010     DAA_issuerSettings -> DAA_generic_q.
011     f. Shift f right by DAA_power0 bits (discard the lowest DAA_power0 bits) and label the
012     result f1
013     g. Set s1 = r1 + (DAA_session -> DAA_digest)* f1 in Z
014     h. set outputData = s1
015     i. increment DAA_session -> DAA_stage by 1
016     j. return TPM_SUCCESS
017 19.If stage==19
018     a. Verify that DAA_session ->DAA_stage==19. Return TPM_DAA_STAGE and flush
019     handle on mismatch
020     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
021     return error TPM_DAA_ISSUER_SETTINGS on mismatch

```

```

022     c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
023     DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
024     d. obtain DAA_SIZE_r2 bits from MGF1("r2", DAA_session -> DAA_contextSeed), and
025     label them r2
026     e. Set s2 = r2 + (DAA_session -> DAA_digest)*( DAA_joinSession -> DAA_join_u0) mod
027     2^DAA_power1 (Erase all but the lowest DAA_power1 bits of s2)
028     f. Set DAA_session -> DAA_scratch = s2
029     g. set outputData = s2
030     h. increment DAA_session -> DAA_stage by 1
031     i. return TPM_SUCCESS
032 20.If stage==20
033     a. Verify that DAA_session ->DAA_stage==20. Return TPM_DAA_STAGE and flush
034     handle on mismatch
035     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
036     return error TPM_DAA_ISSUER_SETTINGS on mismatch
037     c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
038     DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
039     d. obtain DAA_SIZE_r2 bits from MGF1("r2", DAA_session -> DAA_contextSeed), and
040     label them r2
041     e. Set s12 = r2 + (DAA_session -> DAA_digest)*( DAA_joinSession -> DAA_join_u0)
042     f. Shift s12 right by DAA_power1 bit (discard the lowest DAA_power1 bits).
043     g. Set DAA_session -> DAA_scratch = s12
044     h. Set outputData = DAA_session -> DAA_digest
045     i. increment DAA_session -> DAA_stage by 1
046     j. return TPM_SUCCESS
047 21.If stage==21
048     a. Verify that DAA_session ->DAA_stage==21. Return TPM_DAA_STAGE and flush
049     handle on mismatch
050     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
051     return error TPM_DAA_ISSUER_SETTINGS on mismatch
052     c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
053     DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
054     d. obtain DAA_SIZE_r3 bits from MGF1("r3", DAA_session -> DAA_contextSeed), and
055     label them r3
056     e. Set s3 = r3 + (DAA_session -> DAA_digest)*( DAA_joinSession -> DAA_join_u1) +
057     (DAA_session -> DAA_scratch).
058     f. Set DAA_session -> DAA_scratch = NULL
059     g. set outputData = s3

```

```

060      h. increment DAA_session -> DAA_stage by 1
061      i. return TPM_SUCCESS
062  22.If stage==22
063      a. Verify that DAA_session ->DAA_stage==22. Return TPM_DAA_STAGE and flush
064      handle on mismatch
065      b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
066      return error TPM_DAA_ISSUER_SETTINGS on mismatch
067      c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
068      DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
069      d. Verify inputSize0 == DAA_SIZE_v0 and return error TPM_DAA_INPUT_DATA0 on
070      mismatch
071      e. Set u2 = inputData0
072      f. Set v0 = u2 + (DAA_joinSession -> DAA_join_u0) mod 2^DAA_power1 (Erase all but
073      the lowest DAA_power1 bits of v0).
074      g. Set DAA_tpmSpecific -> DAA_digest_v0 = SHA-1(v0)
075      h. Set v10 = u2 + (DAA_joinSession -> DAA_join_u0) in  $\mathbb{Z}$ 
076      i. Shift v10 right by DAA_power1 bits (erase the lowest DAA_power1 bits).
077      j. Set DAA_session ->DAA_scratch = v10
078      k. Set outputData
079          i. Fill in TPM_DAA_BLOB with a type of TPM_RT_DAA_V0 and encrypt the v0
080          parameters
081          ii. set outputData to the encrypted TPM_DAA_BLOB
082      l. increment DAA_session -> DAA_stage by 1
083      m. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific ||
084      DAA_joinSession)
085      n. return TPM_SUCCESS
086  23.If stage==23
087      a. Verify that DAA_session ->DAA_stage==23. Return TPM_DAA_STAGE and flush
088      handle on mismatch
089      b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
090      return error TPM_DAA_ISSUER_SETTINGS on mismatch
091      c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
092      DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
093      d. Verify inputSize0 == DAA_SIZE_v1 and return error TPM_DAA_INPUT_DATA0 on
094      mismatch
095      e. Set u3 = inputData0
096      f. Set v1 = u3 + DAA_joinSession -> DAA_join_u1 + DAA_session ->DAA_scratch
097      g. Set DAA_tpmSpecific -> DAA_digest_v1 = SHA-1(v1)

```

```
098     h. Set outputData
099         i. Fill in TPM_DAA_BLOB with a type of TPM_RT_DAA_V1 and encrypt the v1
100            parameters
101         ii. set outputData to the encrypted TPM_DAA_BLOB
102     i. Set DAA_session ->DAA_scratch = NULL
103     j. increment DAA_session -> DAA_stage by 1
104     k. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific ||
105        DAA_joinSession)
106     l. return TPM_SUCCESS
107 24.If stage==24
108     a. Verify that DAA_session ->DAA_stage==24. Return TPM_DAA_STAGE and flush
109        handle on mismatch
110     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
111        return error TPM_DAA_ISSUER_SETTINGS on mismatch
112     c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
113        DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
114     d. set outputData = enc(DAA_tpmSpecific)
115     e. return TPM_SUCCESS
116 25.If stage > 24, return error: TPM_DAA_STAGE
```

**26.2 TPM\_DAA\_Sign**

TPM protected capability; user must provide authorizations from the TPM Owner.

**Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	Tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes incl. paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	Ordinal	Command ordinal: TPM_ORD_DAA_Sign
4	4			TPM_HANDLE	handle	Handle to the sign session
5	1	2S	1	BYTE	stage	Stage of the sign process
6	4	3S	4	UINT32	inputSize0	Size of inputData0 for this stage of DAA_Sign
7	⇔	4S	⇔	BYTE[]	inputData0	Data to be used by this capability
8	4	5S	4	UINT32	inputSize1	Size of inputData1 for this stage of DAA_Sign
9	⇔	6S	⇔	BYTE[]	inputData1	Data to be used by this capability
10	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication
		2 H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
11	20	3 H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
12	1	4 H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
13	20		20	TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner. HMAC key: ownerAuth.

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes incl. paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DAA_Sign
4	4	3S	4	UINT32	outputSize	Size of outputData
5	⇔	4S	⇔	BYTE[]	outputData	Data produced by this capability
6	20	2 H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3 H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4 H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20		20	TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

## Description

This table summaries the input, output and saved data that is associated with each stage of processing.

Stage	Input Data0	Input Data1	Operation	Output Data	Scratchpad
0	DAA_issuerSettings	NULL	Initialise	handle	NULL
1	enc(DAA_tpmSpecific)	NULL	initialise	NULL	NULL
2	DAA_generic_R0	DAA_generic_n	$P1 = R0 \cdot r0 \bmod n$	NULL	P1
3	DAA_generic_R1	DAA_generic_n	$P2 = P1 \cdot (R1 \cdot r1) \bmod n$	NULL	P2
4	DAA_generic_S0	DAA_generic_n	$P3 = P2 \cdot (S0 \cdot r2) \bmod n$	NULL	P3
5	DAA_generic_S1	DAA_generic_n	$T = P3 \cdot (S1 \cdot r4) \bmod n$	T	NULL
6	DAA_generic_gamma	w	$w1 = w^q \bmod \text{gamma}$	NULL	w
7	DAA_generic_gamma	NULL	$E = w^f \bmod \text{gamma}$	E	w
8	DAA_generic_gamma	NULL	$r = r0 + (2^{\text{power}0}) \cdot r1 \bmod q$ $E1 = w^r \bmod \text{gamma}$	E1	NULL
9	c1	NULL	$c = \text{hash}(c1 \parallel \text{NT})$	NT	NULL
10	b (selector)	m or handle to AIK	$c = \text{hash}(c \parallel 1 \parallel m)$ or $c = \text{hash}(c \parallel 0 \parallel \text{AIK-modulus})$	c	NULL
11	NULL	NULL	$s0 = r0 + c \cdot f0$	s0	NULL
12	NULL	NULL	$s1 = r1 + c \cdot f1$	s1	NULL
13	enc(v0)	NULL	$s2 = r2 + c \cdot v0 \bmod 2^{\text{power}1}$	s2	NULL
14	enc(v0)	NULL	$s12 = r2 + c \cdot v0 \gg \text{power}1$	NULL	s12
15	enc(v1)	NULL	$s3 = r4 + c \cdot v1 + s12$	s3	NULL

When a TPM receives an Owner authorized command to input enc(DAA\_tpmSpecific) or enc(v0) or enc(v1), the TPM MUST verify that the TPM created the data and that neither the data nor the TPM's EK has been changed since the data was created. Loading one of these wrapped blobs does not require authorization, since correct blobs were created by the TPM under Owner authorization, and unwrapped blobs cannot be used without Owner authorisation. The TPM MUST NOT restrict the number of times that the contents of enc(DAA\_tpmSpecific) or enc(v0) or enc(v1) can be used by the same combination of TPM and Owner that created them..

## Actions

A Trusted Platform Module that receives a valid TPM\_DAA\_Sign command SHALL:

26. Use ownerAuth to verify that the Owner authorized all TPM\_DAA\_Sign input parameters.

27. Any error results in the TPM invalidating all resources associated with the command

28. Constant values of 0 or 1 are 1 byte integers, stages affected are

- a. 7(f), 11(e), 12(e)



- 139 29. Representation of the strings “r0” to “r4” are 2-byte ASCII encodings, stages affected are  
140 a. 2(h), 3(h), 4(h), 5(h), 12(d), 13(f), 14(f), 15(f)

## 141 Stages

- 142 0. If stage==0
- 143 a. Determine that sufficient resources are available to perform a TPM\_DAA\_Sign.
    - 144 i. The TPM MUST support sufficient resources to perform one (1) TPM\_DAA\_Join/  
145 TPM\_DAA\_Sign. The TPM MAY support addition TPM\_DAA\_Join/ TPM\_DAA\_Sign  
146 sessions.
    - 147 ii. The TPM may share internal resources between the DAA operations and other  
148 variable resource requirements:
    - 149 iii. If there are insufficient resources within the stored key pool (and one or more  
150 keys need to be removed to permit the DAA operation to execute) return  
151 TPM\_NOSPACE
    - 152 iv. If there are insufficient resources within the stored session pool (and one or  
153 more authorization or transport sessions need to be removed to permit the  
154 DAA operation to execute), return TPM\_RESOURCES.
  - 155 b. Set DAA\_issuerSettings = inputData0
  - 156 c. Verify that all fields in DAA\_issuerSettings are present and return error  
157 TPM\_DAA\_INPUT\_DATA0 if not.
  - 158 d. set all fields in DAA\_session = NULL
  - 159 e. Assign new handle for session
  - 160 f. Set outputData to new handle
  - 161 g. set DAA\_session -> DAA\_stage = 1
  - 162 h. return TPM\_SUCCESS
- 163 1. If stage==1
- 164 a. Verify that DAA\_session ->DAA\_stage==1. Return TPM\_DAA\_STAGE and flush handle  
165 on mismatch
  - 166 b. Set DAA\_tpmSpecific = unwrap(inputData0)
  - 167 c. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
168 return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch
  - 169 d. set DAA\_session -> DAA\_digestContext = SHA-1(DAA\_tpmSpecific)
  - 170 e. obtain random data from the RNG and store it as DAA\_session -> DAA\_contextSeed
  - 171 f. set outputData = NULL
  - 172 g. set DAA\_session -> DAA\_stage =2
  - 173 h. return TPM\_SUCCESS
- 174 2. If stage==2

175       a. Verify that DAA\_session ->DAA\_stage==2. Return TPM\_DAA\_STAGE and flush handle  
176       on mismatch

177       b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
178       return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch

179       c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific) and  
180       return error TPM\_DAA\_TPM\_SETTINGS on mismatch

181       d. Set DAA\_generic\_R0 = inputData0

182       e. Verify that SHA-1(DAA\_generic\_R0) == DAA\_issuerSettings -> DAA\_digest\_R0 and  
183       return error TPM\_DAA\_INPUT\_DATA0 on mismatch

184       f. Set DAA\_generic\_n = inputData1

185       g. Verify that SHA-1(DAA\_generic\_n) == DAA\_issuerSettings -> DAA\_digest\_n and  
186       return error TPM\_DAA\_INPUT\_DATA1 on mismatch

187       h. obtain DAA\_SIZE\_r0 bits from MGF1("r0", DAA\_session -> DAA\_contextSeed), and  
188       label them Y

189       i. Set X = DAA\_generic\_R0

190       j. Set n = DAA\_generic\_n

191       k. Set DAA\_session -> DAA\_scratch = (X^Y) mod n

192       l. set outputData = NULL

193       m. increment DAA\_session -> DAA\_stage by 1

194       n. return TPM\_SUCCESS

195    3. If stage==3

196       a. Verify that DAA\_session ->DAA\_stage==3. Return TPM\_DAA\_STAGE and flush handle  
197       on mismatch

198       b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
199       return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch

200       c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific) and  
201       return error TPM\_DAA\_TPM\_SETTINGS on mismatch

202       d. Set DAA\_generic\_R1 = inputData0

203       e. Verify that SHA-1(DAA\_generic\_R1) == DAA\_issuerSettings -> DAA\_digest\_R1 and  
204       return error TPM\_DAA\_INPUT\_DATA0 on mismatch

205       f. Set DAA\_generic\_n = inputData1

206       g. Verify that SHA-1(DAA\_generic\_n) == DAA\_issuerSettings -> DAA\_digest\_n and  
207       return error TPM\_DAA\_INPUT\_DATA1 on mismatch

208       h. obtain DAA\_SIZE\_r1 bits from MGF1("r1", DAA\_session -> DAA\_contextSeed), and  
209       label them Y

210       i. Set X = DAA\_generic\_R1

211       j. Set n = DAA\_generic\_n

212       k. Set Z = DAA\_session -> DAA\_scratch

213        1. Set DAA\_session -> DAA\_scratch =  $Z*(X^Y) \bmod n$   
214        m. set outputData = NULL  
215        n. increment DAA\_session -> DAA\_stage by 1  
216        o. return TPM\_SUCCESS  
217    4. If stage==4  
218        a. Verify that DAA\_session ->DAA\_stage==4. Return TPM\_DAA\_STAGE and flush handle  
219        on mismatch  
220        b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
221        return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch  
222        c. Verify that DAA\_session -> DAA\_digestContext = SHA-1(DAA\_tpmSpecific) and  
223        return error TPM\_DAA\_TPM\_SETTINGS on mismatch  
224        d. Set DAA\_generic\_S0 = inputData0  
225        e. Verify that SHA-1(DAA\_generic\_S0) == DAA\_issuerSettings -> DAA\_digest\_S0 and  
226        return error TPM\_DAA\_INPUT\_DATA0 on mismatch  
227        f. Set DAA\_generic\_n = inputData1  
228        g. Verify that SHA-1(DAA\_generic\_n) == DAA\_issuerSettings -> DAA\_digest\_n and  
229        return error TPM\_DAA\_INPUT\_DATA1 on mismatch  
230        h. obtain DAA\_SIZE\_r2 bits from MGF1("r2", DAA\_session -> DAA\_contextSeed), and  
231        label them Y  
232        i. Set X = DAA\_generic\_S0  
233        j. Set n = DAA\_generic\_n  
234        k. Set Z = DAA\_session -> DAA\_scratch  
235        l. Set DAA\_session -> DAA\_scratch =  $Z*(X^Y) \bmod n$   
236        m. set outputData = NULL  
237        n. increment DAA\_session -> DAA\_stage by 1  
238        o. return TPM\_SUCCESS  
239    5. If stage==5  
240        a. Verify that DAA\_session ->DAA\_stage==5. Return TPM\_DAA\_STAGE and flush handle  
241        on mismatch  
242        b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
243        return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch  
244        c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific) and  
245        return error TPM\_DAA\_TPM\_SETTINGS on mismatch  
246        d. Set DAA\_generic\_S1 = inputData0  
247        e. Verify that SHA-1(DAA\_generic\_S1) == DAA\_issuerSettings -> DAA\_digest\_S1 and  
248        return error TPM\_DAA\_INPUT\_DATA0 on mismatch  
249        f. Set DAA\_generic\_n = inputData1

```

250     g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and
251     return error TPM_DAA_INPUT_DATA1 on mismatch
252     h. obtain DAA_SIZE_r4 bits from MGF1("r4", DAA_session -> DAA_contextSeed), and
253     label them Y
254     i. Set X = DAA_generic_S1
255     j. Set n = DAA_generic_n
256     k. Set Z = DAA_session -> DAA_scratch
257     l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n
258     m. set outputData = DAA_session -> DAA_scratch
259     n. set DAA_session -> DAA_scratch = NULL
260     o. increment DAA_session -> DAA_stage by 1
261     p. return TPM_SUCCESS
262 6. If stage==6
263     a. Verify that DAA_session ->DAA_stage==6. Return TPM_DAA_STAGE and flush handle
264     on mismatch
265     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
266     return error TPM_DAA_ISSUER_SETTINGS on mismatch
267     c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and
268     return error TPM_DAA_TPM_SETTINGS on mismatch
269     d. Set DAA_generic_gamma = inputData0
270     e. Verify that SHA-1(DAA_generic_gamma) == DAA_issuerSettings ->
271     DAA_digest_gamma and return error TPM_DAA_INPUT_DATA0 on mismatch
272     f. Verify that inputSize1 == DAA_SIZE_w and return error TPM_DAA_INPUT_DATA1 on
273     mismatch
274     g. Set w = inputData1
275     h. Set w1 = w^( DAA_issuerSettings -> DAA_generic_q) mod (DAA_generic_gamma)
276     i. If w1 != 1 (unity), return error TPM_DAA_WRONG_W
277     j. Set DAA_session -> DAA_scratch = w
278     k. set outputData = NULL
279     l. increment DAA_session -> DAA_stage by 1
280     m. return TPM_SUCCESS.
281 7. If stage==7
282     a. Verify that DAA_session ->DAA_stage==7. Return TPM_DAA_STAGE and flush handle
283     on mismatch
284     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
285     return error TPM_DAA_ISSUER_SETTINGS on mismatch

```

```

286 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and
287 return error TPM_DAA_TPM_SETTINGS on mismatch
288 d. Set DAA_generic_gamma = inputData0
289 e. Verify that SHA-1(DAA_generic_gamma) == DAA_issuerSettings ->
290 DAA_digest_gamma and return error TPM_DAA_INPUT_DATA0 on mismatch
291 f. Set f = SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0
292 ) || SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1) mod
293 DAA_issuerSettings -> DAA_generic_q.
294 g. Set E = ((DAA_session -> DAA_scratch)^f) mod (DAA_generic_gamma).
295 h. Set outputData = E
296 i. increment DAA_session -> DAA_stage by 1
297 j. return TPM_SUCCESS.
298 8. If stage==8
299 a. Verify that DAA_session ->DAA_stage==8. Return TPM_DAA_STAGE and flush handle
300 on mismatch
301 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
302 return error TPM_DAA_ISSUER_SETTINGS on mismatch
303 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and
304 return error TPM_DAA_TPM_SETTINGS on mismatch
305 d. Set DAA_generic_gamma = inputData0
306 e. Verify that SHA-1(DAA_generic_gamma) == DAA_issuerSettings ->
307 DAA_digest_gamma and return error TPM_DAA_INPUT_DATA0 on mismatch
308 f. obtain DAA_SIZE_r0 bits from MGF1("r0", DAA_session -> DAA_contextSeed), and
309 label them r0
310 g. obtain DAA_SIZE_r1 bits from MGF1("r1", DAA_session -> DAA_contextSeed), and
311 label them r1
312 h. set r = r0 + 2^DAA_power0 * r1 mod (DAA_issuerSettings -> DAA_generic_q).
313 i. Set E1 = ((DAA_session -> DAA_scratch)^r) mod (DAA_generic_gamma)
314 j. Set DAA_session -> DAA_scratch = NULL
315 k. Set outputData = E1
316 l. increment DAA_session -> DAA_stage by 1
317 m. return TPM_SUCCESS.
318 9. If stage==9
319 a. Verify that DAA_session ->DAA_stage==9. Return TPM_DAA_STAGE and flush handle
320 on mismatch
321 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
322 return error TPM_DAA_ISSUER_SETTINGS on mismatch

```

323 c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific) and  
324 return error TPM\_DAA\_TPM\_SETTINGS on mismatch

325 d. Verify that inputSize0 == sizeof(TPM\_DIGEST) and return error  
326 TPM\_DAA\_INPUT\_DATA0 on mismatch

327 e. Set DAA\_session -> DAA\_digest = inputData0

328 f. obtain DAA\_SIZE\_NT bits from the RNG and label them NT

329 g. Set DAA\_session -> DAA\_digest to the SHA-1 ( DAA\_session -> DAA\_digest || NT )

330 h. Set outputData = NT

331 i. increment DAA\_session -> DAA\_stage by 1

332 j. return TPM\_SUCCESS.

333 10.If stage==10

334 a. Verify that DAA\_session ->DAA\_stage==10. Return TPM\_DAA\_STAGE and flush  
335 handle on mismatch

336 b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
337 return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch

338 c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific) and  
339 return error TPM\_DAA\_TPM\_SETTINGS on mismatch

340 d. Set selector = inputData0, verify that selector == 0 or 1, and return error  
341 TPM\_DAA\_INPUT\_DATA0 on mismatch

342 e. If selector == 1, verify that inputSize1 == sizeof(TPM\_DIGEST), and

343 f. Set DAA\_session -> DAA\_digest to SHA-1 (DAA\_session -> DAA\_digest || 1 ||  
344 inputData1)

345 g. If selector == 0, verify that inputData1 is a handle to a TPM identity key (AIK), and

346 h. Set DAA\_session -> DAA\_digest to SHA-1 (DAA\_session -> DAA\_digest || 0 || n2)  
347 where n2 is the modulus of the AIK

348 i. Set outputData = DAA\_session -> DAA\_digest

349 j. increment DAA\_session -> DAA\_stage by 1

350 k. return TPM\_SUCCESS.

351 11.If stage==11

352 a. Verify that DAA\_session ->DAA\_stage==11. Return TPM\_DAA\_STAGE and flush  
353 handle on mismatch

354 b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
355 return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch

356 c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific) and  
357 return error TPM\_DAA\_TPM\_SETTINGS on mismatch

358 d. obtain DAA\_SIZE\_r0 bits from MGF1("r0", DAA\_session -> DAA\_contextSeed), and  
359 label them r0

```

360     e. Set  $f = \text{SHA1}(\text{DAA\_tpmSpecific} \rightarrow \text{DAA\_rekey} || \text{DAA\_tpmSpecific} \rightarrow \text{DAA\_count} || 0$ 
361  $) || \text{SHA1}(\text{DAA\_tpmSpecific} \rightarrow \text{DAA\_rekey} || \text{DAA\_tpmSpecific} \rightarrow \text{DAA\_count} || 1) \bmod$ 
362  $\text{DAA\_issuerSettings} \rightarrow \text{DAA\_generic\_q}$ 
363     f. Set  $f_0 = f \bmod 2^{\text{DAA\_power0}}$  (erase all but the lowest DAA_power0 bits of f)
364     g. Set  $s_0 = r_0 + (\text{DAA\_session} \rightarrow \text{DAA\_digest}) * (f_0)$ 
365     h. set outputData = s0
366     i. increment DAA_session  $\rightarrow$  DAA_stage by 1
367     j. return TPM_SUCCESS
368 12.If stage==12
369     a. Verify that DAA_session  $\rightarrow$  DAA_stage==12. Return TPM_DAA_STAGE and flush
370     handle on mismatch
371     b. Verify that DAA_tpmSpecific  $\rightarrow$  DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
372     return error TPM_DAA_ISSUER_SETTINGS on mismatch
373     c. Verify that DAA_session  $\rightarrow$  DAA_digestContext == SHA-1(DAA_tpmSpecific) and
374     return error TPM_DAA_TPM_SETTINGS on mismatch
375     d. obtain DAA_SIZE_r1 bits from MGF1("r1", DAA_session  $\rightarrow$  DAA_contextSeed), and
376     label them r1
377     e. Set  $f = \text{SHA1}(\text{DAA\_tpmSpecific} \rightarrow \text{DAA\_rekey} || \text{DAA\_tpmSpecific} \rightarrow \text{DAA\_count} || 0$ 
378  $) || \text{SHA1}(\text{DAA\_tpmSpecific} \rightarrow \text{DAA\_rekey} || \text{DAA\_tpmSpecific} \rightarrow \text{DAA\_count} || 1) \bmod$ 
379  $\text{DAA\_issuerSettings} \rightarrow \text{DAA\_generic\_q}$ 
380     f. Shift f right by DAA_power0 bits (discard the lowest DAA_power0 bits) and label the
381     result f1
382     g. Set  $s_1 = r_1 + (\text{DAA\_session} \rightarrow \text{DAA\_digest}) * (f_1)$ 
383     h. set outputData = s1
384     i. increment DAA_session  $\rightarrow$  DAA_stage by 1
385     j. return TPM_SUCCESS
386 13.If stage==13
387     a. Verify that DAA_session  $\rightarrow$  DAA_stage==13. Return TPM_DAA_STAGE and flush
388     handle on mismatch
389     b. Verify that DAA_tpmSpecific  $\rightarrow$  DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
390     return error TPM_DAA_ISSUER_SETTINGS on mismatch
391     c. Verify that DAA_session  $\rightarrow$  DAA_digestContext == SHA-1(DAA_tpmSpecific) and
392     return error TPM_DAA_TPM_SETTINGS on mismatch
393     d. Set DAA_private_v0 = unwrap(inputData0)
394     e. Verify that SHA-1(DAA_private_v0) == DAA_tpmSpecific  $\rightarrow$  DAA_digest_v0 and return
395     error TPM_DAA_INPUT_DATA0 on mismatch
396     f. obtain DAA_SIZE_r2 bits from MGF1("r2", DAA_session  $\rightarrow$  DAA_contextSeed), and
397     label them r2

```

```

398     g. Set  $s2 = r2 + (DAA\_session \rightarrow DAA\_digest) * (DAA\_private\_v0) \bmod 2^{DAA\_power1}$ 
399     (erase all but the lowest  $DAA\_power1$  bits of  $s2$ )
400     h. Set  $DAA\_session \rightarrow DAA\_scratch = s2$ 
401     i. set outputData = s2
402     j. increment  $DAA\_session \rightarrow DAA\_stage$  by 1
403     k. return TPM_SUCCESS
404 14.If stage==14
405     a. Verify that  $DAA\_session \rightarrow DAA\_stage == 1$ . Return TPM_DAA_STAGE and flush handle
406     on mismatch
407     b. Verify that  $DAA\_tpmSpecific \rightarrow DAA\_digestIssuer == SHA-1(DAA\_issuerSettings)$  and
408     return error TPM_DAA_ISSUER_SETTINGS on mismatch
409     c. Verify that  $DAA\_session \rightarrow DAA\_digestContext == SHA-1(DAA\_tpmSpecific)$  and
410     return error TPM_DAA_TPM_SETTINGS on mismatch
411     d. Set  $DAA\_private\_v0 = unwrap(inputData0)$ 
412     e. Verify that  $SHA-1(DAA\_private\_v0) == DAA\_tpmSpecific \rightarrow DAA\_digest\_v0$  and return
413     error TPM_DAA_INPUT_DATA0 on mismatch
414     f. obtain  $DAA\_SIZE\_r2$  bits from  $MGF1("r2", DAA\_session \rightarrow DAA\_contextSeed)$ , and
415     label them  $r2$ 
416     g. Set  $s12 = r2 + (DAA\_session \rightarrow DAA\_digest) * (DAA\_private\_v0)$ .
417     h. Shift  $s12$  right by  $DAA\_power1$  bits (erase the lowest  $DAA\_power1$  bits).
418     i. Set  $DAA\_session \rightarrow DAA\_scratch = s12$ 
419     j. set outputData = NULL
420     k. increment  $DAA\_session \rightarrow DAA\_stage$  by 1
421     l. return TPM_SUCCESS
422 15.If stage==15
423     a. Verify that  $DAA\_session \rightarrow DAA\_stage == 15$ . Return TPM_DAA_STAGE and flush
424     handle on mismatch
425     b. Verify that  $DAA\_tpmSpecific \rightarrow DAA\_digestIssuer == SHA-1(DAA\_issuerSettings)$  and
426     return error TPM_DAA_ISSUER_SETTINGS on mismatch
427     c. Verify that  $DAA\_session \rightarrow DAA\_digestContext == SHA-1(DAA\_tpmSpecific)$  and
428     return error TPM_DAA_TPM_SETTINGS on mismatch
429     d. Set  $DAA\_private\_v1 = unwrap(inputData0)$ 
430     e. Verify that  $SHA-1(DAA\_private\_v1) == DAA\_tpmSpecific \rightarrow DAA\_digest\_v1$  and return
431     error TPM_DAA_INPUT_DATA0 on mismatch
432     f. obtain  $DAA\_SIZE\_r4$  bits from  $MGF1("r4", DAA\_session \rightarrow DAA\_contextSeed)$ , and
433     label them  $r4$ 
434     g. Set  $s3 = r4 + (DAA\_session \rightarrow DAA\_digest) * (DAA\_private\_v1) + (DAA\_session \rightarrow$ 
435      $DAA\_scratch)$ .

```



436       h. Set DAA\_session -> DAA\_scratch = NULL  
437       i. set outputData = s3  
438       j. increment DAA\_session -> DAA\_stage by 1  
439       k. return TPM\_SUCCESS  
440   16.If stage > 15, return error: TPM\_DAA\_STAGE

## 27. Deprecated commands

### **Start of informative comment:**

This section covers the commands that were in version 1.1 but now have new functionality in other functions. The deprecated commands are still available in 1.2 but all new software should use the new functionality.

There is no requirement that the deprecated commands work with new structures.

### **End of informative comment.**

1. Commands deprecated in version 1.2 MUST work with version 1.1 structures
2. Commands deprecated in version 1.2 MAY work with version 1.2 structures

**27.1 Key commands****Start of informative comment:**

The key commands are deprecated as the new way to handle keys is to use the standard context commands. So TPM\_EvictKey is now handled by TPM\_FlushSpecific, TPM\_Terminate\_Handle by TPM\_FlushSpecific.

**End of informative comment.****27.1.1 TPM\_EvictKey****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_EvictKey
4	4			TPM_KEY_HANDLE	evictHandle	The handle of the key to be evicted.

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_EvictKey

**Actions**

The TPM will invalidate the key stored in the specified handle and return the space to the available internal pool for subsequent query by TPM\_GetCapability and usage by TPM\_LoadKey. If the specified key handle does not correspond to a valid key, an error will be returned.

**New 1.2 functionality**

The command must check the status of the ownerEvict flag for the key and if the flag is TRUE return TPM\_KEY\_CONTROL\_OWNER

## 27.1.2 TPM\_Terminate\_Handle

### Start of informative comment:

This allows the TPM manager to clear out information in a session handle.

The TPM may maintain the authorization session even though a key attached to it has been unloaded or the authorization session itself has been unloaded in some way. When a command is executed that requires this session, it is the responsibility of the external software to load both the entity and the authorization session information prior to command execution.

### End of informative comment.

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Terminate_Handle.
4	4			TPM_AUTHHANDLE	handle	The handle to terminate

### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Terminate_Handle.

### Descriptions

The TPM SHALL terminate the session and destroy all data associated with the session indicated.

### Actions

A TPM SHALL unilaterally perform the actions of TPM\_Terminate\_Handle upon detection of the following events:

1. Completion of a received command whose authorization "continueUse" flag is FALSE.
2. Completion of a received command when a shared secret derived from the authorization session was exclusive-or'ed with data (to provide confidentiality for that data). This occurs during execution of a TPM\_ChangeAuth command, for example.
3. When the associated entity is destroyed (in the case of TPM Owner or SRK, for example)
4. Upon execution of TPM\_Init

- 490 5. When the command returns an error. This is due to the fact that when returning an  
491 error the TPM does not send back nonceEven. There is no way to maintain the rolling  
492 nonces, hence the TPM MUST terminate the authorization session.
- 493 6. Failure of an authorization check belonging to that authorization session.

## 27.2 Context management

### Start of informative comment:

The 1.1 context commands were written for specific resource types. The 1.2 commands are generic for all resource types. So the Savexxx commands are replaced by TPM\_SaveContext and the LoadXXX commands by TPM\_LoadContext.

### End of informative comment.

## 27.2.1 TPM\_SaveKeyContext

### Start of informative comment:

TPM\_SaveKeyContext saves a loaded key outside the TPM. After creation of the key context blob the TPM automatically releases the internal memory used by that key. The format of the key context blob is specific to a TPM.

### End of informative comment.

## Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveKeyContext
4	4			TPM_KEY_HANDLE	keyHandle	The key which will be kept outside the TPM

## Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveKeyContext
4	4	3S	4	UINT32	keyContextSize	The actual size of the outgoing key context blob. If the command fails the value will be 0
5	<>	4S	<>	BYTE[]	keyContextBlob	The key context blob.

## Description

1. This command allows saving a loaded key outside the TPM. After creation of the keyContextBlob, the TPM automatically releases the internal memory used by that key. The format of the key context blob is specific to a TPM.
2. A TPM protected capability belonging to the TPM that created a key context blob MUST be the only entity that can interpret the contents of that blob. If a cryptographic technique is used for this purpose, the level of security provided by that technique SHALL be at least as secure as a 2048 bit RSA algorithm. Any secrets (such as keys)

- 516        used in such a cryptographic technique MUST be generated using the TPM's random  
517        number generator. Any symmetric key MUST be used within the power-on session  
518        during which it was created, only.
- 519        3. A key context blob SHALL enable verification of the integrity of the contents of the blob  
520        by a TPM protected capability.
- 521        4. A key context blob SHALL enable verification of the session validity of the contents of the  
522        blob by a TPM protected capability. The method SHALL ensure that all key context blobs  
523        are rendered invalid if power to the TPM is interrupted.

## 27.2.2 TPM\_LoadKeyContext

### Start of informative comment:

TPM\_LoadKeyContext loads a key context blob into the TPM previously retrieved by a TPM\_SaveKeyContext call. After successful completion the handle returned by this command can be used to access the key.

### End of informative comment.

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadKeyContext
4	4	2S	4	UINT32	keyContextSize	The size of the following key context blob.
5	<>	3S	<>	BYTE[]	keyContextBlob	The key context blob.

### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadKeyContext
4	4			TPM_KEY_HANDLE	keyHandle	The handle assigned to the key after it has been successfully loaded.

### Description

1. This command allows loading a key context blob into the TPM previously retrieved by a TPM\_SaveKeyContext call. After successful completion the handle returned by this command can be used to access the key.
2. The contents of a key context blob SHALL be discarded unless the contents have passed an integrity test. This test SHALL (statistically) prove that the contents of the blob are the same as when the blob was created.
3. The contents of a key context blob SHALL be discarded unless the contents have passed a session validity test. This test SHALL (statistically) prove that the blob was created by this TPM during this power-on session.



### 27.2.3 TPM\_SaveAuthContext

#### Start of informative comment:

TPM\_SaveAuthContext saves a loaded authorization session outside the TPM. After creation of the authorization context blob, the TPM automatically releases the internal memory used by that session. The format of the authorization context blob is specific to a TPM.

#### End of informative comment.

#### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveAuthContext
4	4			TPM_AUTHHANDLE	authHandle	Authorization session which will be kept outside the TPM

#### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveAuthContext
4	4	3S	4	UINT32	authContextSize	The actual size of the outgoing authorization context blob. If the command fails the value will be 0.
5	◇	4S	4	BYTE[]	authContextBlob	The authorization context blob.

#### Description

This command allows saving a loaded authorization session outside the TPM. After creation of the authContextBlob, the TPM automatically releases the internal memory used by that session. The format of the authorization context blob is specific to a TPM.

A TPM protected capability belonging to the TPM that created an authorization context blob MUST be the only entity that can interpret the contents of that blob. If a cryptographic technique is used for this purpose, the level of security provided by that technique SHALL be at least as secure as a 2048 bit RSA algorithm. Any secrets (such as keys) used in such a cryptographic technique MUST be generated using the TPM's random number generator. Any symmetric key MUST be used within the power-on session during which it was created, only.

An authorization context blob SHALL enable verification of the integrity of the contents of the blob by a TPM protected capability.

An authorization context blob SHALL enable verification of the session validity of the contents of the blob by a TPM protected capability. The method SHALL ensure that all authorization context blobs are rendered invalid if power to the TPM is interrupted.

## 27.2.4 TPM\_LoadAuthContext

### Start of informative comment:

TPM\_LoadAuthContext loads an authorization context blob into the TPM previously retrieved by a TPM\_SaveAuthContext call. After successful completion the handle returned by this command can be used to access the authorization session.

### End of informative comment.

## Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadAuthContext
4	4	2S	4	UINT32	authContextSize	The size of the following authorization context blob.
5	<	3S	<	BYTE[]	authContextBlob	The authorization context blob.

## Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadAuthContext
4	4			TPM_KEY_HANDLE	authHandle	The handle assigned to the authorization session after it has been successfully loaded.

### Description

This command allows loading an authorization context blob into the TPM previously retrieved by a TPM\_SaveAuthContext call. After successful completion the handle returned by this command can be used to access the authorization session.

The contents of an authorization context blob SHALL be discarded unless the contents have passed an integrity test. This test SHALL (statistically) prove that the contents of the blob are the same as when the blob was created.

The contents of an authorization context blob SHALL be discarded unless the contents have passed a session validity test. This test SHALL (statistically) prove that the blob was created by this TPM during this power-on session.

## 27.3 DIR commands

**Start of informative comment:**

The DIR commands are replaced by the NV storage commands.

The DIR [0] in 1.1 is now TPM\_PERMANENT\_DATA -> authDIR[0] and is always available for the TPM to use. It is accessed by DIR commands using dirIndex 0 and by NV commands using nvIndex TPM\_NV\_INDEX\_DIR.

If the TPM vendor supports additional DIR registers, the TPM vendor may return errors or provide vendor specific mappings for those DIR registers to NV storage locations.

**End of informative comment.**

1. A dirIndex value of 0 MUST corresponds to an NV storage nvIndex value TPM\_NV\_INDEX\_DIR.
2. The TPM vendor MAY return errors or MAY provide vendor specific mappings for DIR dirIndex values greater than 0 to NV storage locations.

## 27.3.1 TPM\_DirWriteAuth

### Start of informative comment:

The TPM\_DirWriteAuth operation provides write access to the Data Integrity Registers. DIRs are non-volatile memory registers held in a TPM-shielded location. Owner authentication is required to authorize this action.

Access is also provided through the NV commands with nvIndex TPM\_NV\_INDEX\_DIR. Owner authorization is not required when nvLocked is FALSE.

Version 1.2 requires only one DIR. If the DIR named does not exist, the TPM\_DirWriteAuth operation returns TPM\_BADINDEX.

### End of informative comment.

## Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DirWriteAuth.
4	4	2S	4	TPM_DIRINDEX	dirIndex	Index of the DIR
5	20	3S	20	TPM_DIRVALUE	newContents	New value to be stored in named DIR
6	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for command.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
7	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
9	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs. HMAC key: ownerAuth.

## Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DirWriteAuth
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

## Actions

- 610 1. Validate that authHandle contains a TPM Owner AuthData to execute the
- 611 TPM\_DirWriteAuth command
- 612 2. Validate that dirIndex points to a valid DIR on this TPM
- 613 3. Write newContents into the DIR pointed to by dirIndex

## 27.3.2 TPM\_DirRead

### Start of informative comment:

The TPM\_DirRead operation provides read access to the DIRs. No authentication is required to perform this action because typically no cryptographically useful AuthData is available early in boot. TSS implementers may choose to provide other means of authorizing this action. Version 1.2 requires only one DIR. If the DIR named does not exist, the TPM\_DirRead operation returns TPM\_BADINDEX.

### End of informative comment.

## Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DirRead.
4	4	2S	4	TPM_DIRINDEX	dirIndex	Index of the DIR to be read

## Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DirRead.
4	20	3S	20	TPM_DIRVALUE	dirContents	The current contents of the named DIR

## Actions

1. Validate that dirIndex points to a valid DIR on this TPM
2. Return the contents of the DIR in dirContents

628 **27.4 Change Auth**629 **Start of informative comment:**

630 The change auth commands can be duplicated by creating a transport session with  
631 confidentiality and issuing the changeAuth command.

632 **End of informative comment.**

## 27.4.1 TPM\_ChangeAuthAsymStart

### Start of informative comment:

The TPM\_ChangeAuthAsymStart starts the process of changing AuthData for an entity. It sets up an OIAP session that must be retained for use by its twin TPM\_ChangeAuthAsymFinish command.

TPM\_ChangeAuthAsymStart creates a temporary asymmetric public key “tempkey” to provide confidentiality for new AuthData to be sent to the TPM. TPM\_ChangeAuthAsymStart certifies that tempkey was generated by a genuine TPM, by generating a certifyInfo structure that is signed by a TPM identity. The owner of that TPM identity must cooperate to produce this command, because TPM\_ChangeAuthAsymStart requires authorization to use that identity.

It is envisaged that tempkey and certifyInfo are given to the owner of the entity whose authorization is to be changed. That owner uses certifyInfo and a TPM\_IDENTITY\_CREDENTIAL to verify that tempkey was generated by a genuine TPM. This is done by verifying the TPM\_IDENTITY\_CREDENTIAL using the public key of a CA, verifying the signature on the certifyInfo structure with the public key of the identity in TPM\_IDENTITY\_CREDENTIAL, and verifying tempkey by comparing its digest with the value inside certifyInfo. The owner uses tempkey to encrypt the desired new AuthData and inserts that encrypted data in a TPM\_ChangeAuthAsymFinish command, in the knowledge that only a TPM with a specific identity can interpret the new AuthData.

### End of informative comment.

## Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ChangeAuthAsymStart.
4	4			TPM_KEY_HANDLE	idHandle	The keyHandle identifier of a loaded identity ID key
5	20	2s	20	TPM_NONCE	antiReplay	The nonce to be inserted into the certifyInfo structure
6	<>	3S	<>	TPM_KEY_PARMS	tempKey	Structure contains all parameters of ephemeral key.
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for idHandle authorization.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
10	20			TPM_AUTHDATA	idAuth	Authorization. HMAC key: idKey.usageAuth.



655 **Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ChangeAuthAsymStart
7	95	3S	95	TPM_CERTIFY_INFO	certifyInfo	The certifyInfo structure that is to be signed.
8	4	4S	4	UINT32	sigSize	The used size of the output area for the signature
9	◇	5S	◇	BYTE[]	sig	The signature of the certifyInfo parameter.
10	4	6s	4	TPM_KEY_HANDLE	ephHandle	The keyHandle identifier to be used by ChangeAuthAsymFinish for the ephemeral key
11	◇	7S	◇	TPM_KEY	tempKey	Structure containing all parameters and public part of ephemeral key. TPM_KEY.encSize is set to 0.
12	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
13	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
14	20			TPM_AUTHDATA	resAuth	Authorization. HMAC key: idKey.usageAuth.

656 **Actions**

- 657 1. The TPM SHALL verify the AuthData to use the TPM identity key held in idHandle. The  
658 TPM MUST verify that the key is a TPM identity key.
- 659 2. The TPM SHALL validate the algorithm parameters for the key to create from the  
660 tempKey parameter.
- 661 3. Recommended key type is RSA
- 662 4. Minimum RSA key size MUST is 512 bits, recommended RSA key size is 1024
- 663 5. For other key types the minimum key size strength MUST be comparable to RSA 512
- 664 6. If the TPM is not designed to create a key of the requested type, return the error code  
665 TPM\_BAD\_KEY\_PROPERTY
- 666 7. The TPM SHALL create a new key (k1) in accordance with the algorithm parameter. The  
667 newly created key is pointed to by ephHandle.
- 668 8. The TPM SHALL fill in all fields in tempKey using k1 for the information. The TPM\_KEY -  
669 > encSize MUST be 0.
- 670 9. The TPM SHALL fill in certifyInfo using k1 for the information. The certifyInfo -> data  
671 field is supplied by the antiReplay.
- 672 10. The TPM then signs the certifyInfo parameter using the key pointed to by idHandle. The  
673 resulting signed blob is returned in sig parameter

674 **Field Descriptions for certifyInfo parameter**

Type	Name	Description
TPM_VERSION	Version	TPM version structure; Part 2 TPM_VERSION
keyFlags	Redirection	This SHALL be set to FALSE
	Migratable	This SHALL be set to FALSE
	Volatile	This SHALL be set to TRUE
TPM_AUTH_DATA_USAGE	authDataUsage	This SHALL be set to TPM_AUTH_NEVER
TPM_KEY_USAGE	KeyUsage	This SHALL be set to TPM_KEY_AUTHCHANGE
UINT32	PCRInfoSize	This SHALL be set to 0
TPM_DIGEST	pubDigest	This SHALL be the hash of the public key being certified.
TPM_NONCE	Data	This SHALL be set to antiReplay
TPM_KEY_PARMS	info	This specifies the type of key and its parameters.
BOOL	parentPCRStatus	This SHALL be set to FALSE.

675

**27.4.2 TPM\_ChangeAuthAsymFinish****Start of informative comment:**

The TPM\_ChangeAuth command allows the owner of an entity to change the AuthData for the entity.

The command requires the cooperation of the owner of the parent of the entity, since AuthData must be provided to use that parent entity. The command requires knowledge of the existing AuthData information and passes the new AuthData information. The newAuthLink parameter proves knowledge of existing AuthData information and new AuthData information. The new AuthData information “encNewAuth” is encrypted using the “tempKey” variable obtained via TPM\_ChangeAuthAsymStart.

A parent therefore retains control over a change in the AuthData of a child, but is prevented from knowing the new AuthData for that child.

The changeProof parameter provides a proof that the new AuthData value was properly inserted into the entity. The inclusion of a nonce from the TPM provides an entropy source in the case where the AuthData value may be in itself be a low entropy value (hash of a password etc).

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ChangeAuthAsymFinish
4	4			TPM_KEY_HANDLE	parentHandle	The keyHandle of the parent key for the input data
5	4			TPM_KEY_HANDLE	ephHandle	The keyHandle identifier for the ephemeral key
6	2	3S	2	TPM_ENTITY_TYPE	entityType	The type of entity to be modified
7	20	4s	20	TPM_HMAC	newAuthLink	HMAC calculation that links the old and new AuthData values together
8	4	5S	4	UINT32	newAuthSize	Size of encNewAuth
9	<>	6S	<>	BYTE[]	encNewAuth	New AuthData encrypted with ephemeral key.
10	4	7S	4	UINT32	encDataSize	The size of the inData parameter
11	<>	8S	<>	BYTE[]	encData	The encrypted entity that is to be modified.
12	4			TPM_AUTHHANDLE	authHandle	Authorization for parent key.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
13	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
14	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
15	20			TPM_AUTHDATA	privAuth	The authorization session digest for inputs and parentHandle. HMAC key: parentKey.usageAuth.

## 695 Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ChangeAuthAsymFinish
4	4	3S	4	UINT32	outDataSize	The used size of the output area for outData
5	◇	4S	◇	BYTE[]	outData	The modified, encrypted entity.
6	20	5s	20	TPM_NONCE	saltNonce	A nonce value from the TPM RNG to add entropy to the changeProof value
7	◇	6S	◇	TPM_DIGEST	changeProof	Proof that AuthData has changed.
8	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
10	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: parentKey.usageAuth.

## 696 Description

697 If the parentHandle points to the SRK then the HMAC key MUST be built using the TPM  
698 Owner authentication.

## 699 Actions

- 700 1. The TPM SHALL validate that the authHandle parameter authorizes use of the key in  
701 parentHandle.
- 702 2. The encData field MUST be the encData field from TPM\_STORED\_DATA or TPM\_KEY.
- 703 3. The TPM SHALL create e1 by decrypting the entity held in the encData parameter.
- 704 4. The TPM SHALL create a1 by decrypting encNewAuth using the ephHandle ->  
705 TPM\_KEY\_AUTHCHANGE private key. a1 is a structure of type  
706 TPM\_CHANGEAUTH\_VALIDATE.
- 707 5. The TPM SHALL create b1 by performing the following HMAC calculation: b1 = HMAC  
708 (a1 -> newAuthSecret). The secret for this calculation is encData -> currentAuth. This  
709 means that b1 is a value built from the current AuthData value (encData ->  
710 currentAuth) and the new AuthData value (a1 -> newAuthSecret).
- 711 6. The TPM SHALL compare b1 with newAuthLink. The TPM SHALL indicate a failure if the  
712 values do not match.
- 713 7. The TPM SHALL replace e1 -> authData with a1 -> newAuthSecret
- 714 8. The TPM SHALL encrypt e1 using the appropriate functions for the entity type. The key  
715 to encrypt with is parentHandle.
- 716 9. The TPM SHALL create saltNonce by taking the next 20 bytes from the TPM RNG.

- 717 10.The TPM SHALL create changeProof a HMAC of (saltNonce concatenated with a1 -> n1)  
718 using a1 -> newAuthSecret as the HMAC secret.
- 719 11.The TPM MUST destroy the TPM\_KEY\_AUTHCHANGE key associated with the  
720 authorization session.

## 27.5 TPM\_Reset

### Start of informative comment:

TPM\_Reset releases all resources associated with existing authorization sessions. This is useful if a TSS driver has lost track of the state in the TPM.

### End of informative comment.

Deprecated Command in 1.2

### Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Reset.

### Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Reset.

### Description

This is a deprecated command in V1.2. This command in 1.1 only referenced authorization sessions and is not upgraded to affect any other TPM entity in 1.2

### Actions

- The TPM invalidates all resources allocated to authorization sessions as per version 1.1 extant in the TPM
  - This includes structures created by TPM\_SaveAuthContext and TPM\_SaveKeyContext
  - Structures created by TPM\_Contextxxx (the new 1.2 commands) are not affected by this command
- The TPM does not reset any PCR or DIR values.
- The TPM does not reset any flags in the TPM\_STCLEAR\_FLAGS structure.
- The TPM does not reset or invalidate any keys

**27.6 TPM\_OwnerReadPubek****Start of informative comment:**

Return the endorsement key public portion. This is authorized by the TPM Owner.

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OwnerReadPubek
4	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
5	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
7	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner authentication. HMAC key: ownerAuth.

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OwnerReadPubek
4	<>	3S	<>	TPM_PUBKEY	pubEndorsementKey	The public endorsement key
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

**Description**

This command returns the PUBEK.

**Actions**

The TPM\_OwnerReadPubek command SHALL

1. Validate the TPM Owner AuthData to execute this command
2. Export the PUBEK

## 27.7 TPM\_DisablePubekRead

### Start of informative comment:

The TPM Owner may wish to prevent any entity from reading the PUBEK. This command sets the non-volatile flag so that the TPM\_ReadPubek command always returns TPM\_DISABLED\_CMD.

This command has in essence been deprecated as TPM\_TakeOwnership now sets the value to false. The command remains at this time for backward compatibility.

### End of informative comment.

### Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DisablePubekRead
4	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
5	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
7	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner authorization. HMAC key: ownerAuth.

### Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DisablePubekRead
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

### Actions

1. This capability sets the TPM\_PERMANENT\_FLAGS -> readPubek flag to FALSE.



## 27.8 TPM\_LoadKey

### Start of informative comment:

Version 1.2 deprecates TPM\_LoadKey due to the HMAC of the new key handle on return. The wrapping makes use of the handle difficult in an environment where the TSS, or other management entity, is changing the TPM handle to a virtual handle.

Software using TPM\_LoadKey on a 1.2 TPM can have a collision with the returned handle as the 1.2 TPM uses random values in the lower three bytes of the handle. All new software must use LoadKey2 to allow management software the ability to manage the key handle.

Before the TPM can use a key to either wrap, unwrap, bind, unbind, seal, unseal, sign or perform any other action, it needs to be present in the TPM. The TPM\_LoadKey function loads the key into the TPM for further use.

The TPM assigns the key handle. The TPM always locates a loaded key by use of the handle. The assumption is that the handle may change due to key management operations. It is the responsibility of upper level software to maintain the mapping between handle and any label used by external software.

This command has the responsibility of enforcing restrictions on the use of keys. For example, when attempting to load a STORAGE key it will be checked for the restrictions on a storage key (2048 size etc.).

The load command must maintain a record of whether any previous key in the key hierarchy was bound to a PCR using parentPCRStatus.

The flag parentPCRStatus enables the possibility of checking that a platform passed through some particular state or states before finishing in the current state. A grandparent key could be linked to state-1, a parent key could be linked to state-2, and a child key could be linked to state-3, for example. The use of the child key then indicates that the platform passed through states 1 and 2 and is currently in state 3, in this example. TPM\_Startup with stType == TPM\_ST\_CLEAR indicates that the platform has been reset, so the platform has not passed through the previous states. Hence keys with parentPCRStatus==TRUE must be unloaded if TPM\_Startup is issued with stType == TPM\_ST\_CLEAR.

If a TPM\_KEY structure has been decrypted AND the integrity test using "pubDataDigest" has passed AND the key is non-migratory, the key must have been created by the TPM. So there is every reason to believe that the key poses no security threat to the TPM. While there is no known attack from a rogue migratory key, there is a desire to verify that a loaded migratory key is a real key, arising from a general sense of unease about execution of arbitrary data as a key. Ideally a consistency check would consist of an encrypt/decrypt cycle, but this may be expensive. For RSA keys, it is therefore suggested that the consistency test consists of dividing the supposed RSA product by the supposed RSA prime, and checking that there is no remainder.

### End of informative comment.

### 803 Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadKey.
4	4			TPM_KEY_HANDLE	parentHandle	TPM handle of parent key.
5	<>	2S	<>	TPM_KEY	inKey	Incoming key structure, both encrypted private and clear public portions. MAY be TPM_KEY12
6	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for parentHandle authorization.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
7	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
9	20			TPM_AUTHDATA	parentAuth	The authorization session digest for inputs and parentHandle. HMAC key: parentKey.usageAuth.

### 804 Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadKey
4	4	3S	4	TPM_KEY_HANDLE	inkeyHandle	Internal TPM handle where decrypted key was loaded.
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: parentKey.usageAuth.

### 805 Actions

- 806 The TPM SHALL perform the following steps:
- 807 1. Validate the command and the parameters using parentAuth and parentHandle ->  
808 usageAuth
  - 809 2. If parentHandle -> keyUsage is NOT TPM\_KEY\_STORAGE return  
810 TPM\_INVALID\_KEYUSAGE
  - 811 3. If the TPM is not designed to operate on a key of the type specified by inKey, return the  
812 error code TPM\_BAD\_KEY\_PROPERTY
  - 813 4. The TPM MUST handle both TPM\_KEY and TPM\_KEY12 structures
  - 814 5. Decrypt the inKey -> privkey to obtain TPM\_STORE\_ASYMKEY structure using the key  
815 in parentHandle

- 816 6. Validate the integrity of inKey and decrypted TPM\_STORE\_ASYMKEY
- 817 a. Reproduce inKey -> TPM\_STORE\_ASYMKEY -> pubDataDigest using the fields of
- 818 inKey, and check that the reproduced value is the same as pubDataDigest
- 819 7. Validate the consistency of the key and it's key usage.
- 820 a. If inKey -> keyFlags -> migratable is TRUE, the TPM SHALL verify consistency of the
- 821 public and private components of the asymmetric key pair. If inKey -> keyFlags ->
- 822 migratable is FALSE, the TPM MAY verify consistency of the public and private
- 823 components of the asymmetric key pair. The consistency of an RSA key pair MAY be
- 824 verified by dividing the supposed (P\*Q) product by a supposed prime and checking that
- 825 there is no remainder..
- 826 b. If inKey -> keyUsage is TPM\_KEY\_IDENTITY, verify that inKey->keyFlags->migratable
- 827 is FALSE. If it is not, return TPM\_INVALID\_KEYUSAGE
- 828 c. If inKey -> keyUsage is TPM\_KEY\_AUTHCHANGE, return TPM\_INVALID\_KEYUSAGE
- 829 d. If inKey -> keyFlags -> migratable equals 0 then verify that TPM\_STORE\_ASYMKEY -
- 830 > migrationAuth equals TPM\_PERMANENT\_DATA -> tpmProof
- 831 e. Validate the mix of encryption and signature schemes
- 832 f. If TPM\_PERMANENT\_FLAGS -> FIPS is TRUE then
- 833 i. If keyInfo -> keySize is less than 1024 return TPM\_NOTFIPS
- 834 ii. If keyInfo -> authDataUsage specifies TPM\_AUTH\_NEVER return TPM\_NOTFIPS
- 835 iii. If keyInfo -> keyUsage specifies TPM\_KEY\_LEGACY return TPM\_NOTFIPS
- 836 g. If inKey -> keyUsage is TPM\_KEY\_STORAGE or TPM\_KEY\_MIGRATE
- 837 i. algorithmID MUST be TPM\_ALG\_RSA
- 838 ii. Key size MUST be 2048
- 839 iii. sigScheme MUST be TPM\_SS\_NONE
- 840 h. If inKey -> keyUsage is TPM\_KEY\_IDENTITY
- 841 i. algorithmID MUST be TPM\_ALG\_RSA
- 842 ii. Key size MUST be 2048
- 843 iii. encScheme MUST be TPM\_ES\_NONE
- 844 i. If the decrypted inKey -> pcrInfo is NULL,
- 845 i. The TPM MUST set the internal indicator to indicate that the key is not using any
- 846 PCR registers.
- 847 j. Else
- 848 i. The TPM MUST store pcrInfo in a manner that allows the TPM to calculate a
- 849 composite hash whenever the key will be in use
- 850 ii. The TPM MUST handle both version 1.1 TPM\_PCR\_INFO and 1.2
- 851 TPM\_PCR\_INFO\_LONG structures according to the type of TPM\_KEY structure
- 852 iii. The TPM MUST validate the TPM\_PCR\_INFO or TPM\_PCR\_INFO\_LONG
- 853 structures

- 854 8. Perform any processing necessary to make TPM\_STORE\_ASYMKEY key available for  
855 operations
- 856 9. Load key and key information into internal memory of the TPM. If insufficient memory  
857 exists return error TPM\_NOSPACE.
- 858 10. Assign inKeyHandle according to internal TPM rules.
- 859 11. Set InKeyHandle -> parentPCRStatus to parentHandle -> parentPCRStatus.
- 860 12. If ParentHandle indicates it is using PCR registers then set inKeyHandle ->  
861 parentPCRStatus to TRUE.

## 28. Deleted Commands

**Start of informative comment:**

These commands are no longer active commands. Their removal is due to security concerns with their use.

**End of informative comment.**

1. The TPM MUST return TPM\_BAD\_ORDINAL for any deleted command

## 28.1 TPM\_GetCapabilitySigned

### **Start of informative comment:**

Along with TPM\_GetCapabilityOwner this command allowed the possible signature of improper values.

TPM\_GetCapabilitySigned is almost the same as TPM\_GetCapability. The differences are that the input includes a challenge (a nonce) and the response includes a digital signature to vouch for the source of the answer.

If a caller itself requires proof, it is sufficient to use any signing key for which only the TPM and the caller have AuthData.

If a caller requires proof for a third party, the signing key must be one whose signature is trusted by the third party. A TPM-identity key may be suitable.

### **End of informative comment.**

### **Deleted Ordinal**

TPM\_GetCapabilitySigned

**28.2 TPM\_GetOrdinalAuditStatus****Start of informative comment:**

Get the status of the audit flag for the given ordinal.

**End of informative comment.****Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4			TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetOrdinalAuditStatus
4	4			TPM_COMMAND_CODE	ordinalToQuery	The ordinal whose audit flag is to be queried

**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4			TPM_RESULT	returnCode	The return code of the operation.
4	1			BOOL	State	Value of audit flag for ordinalToQuery

**Actions**

1. The TPM returns the Boolean value for the given ordinal. The value is TRUE if the command is being audited.

## 28.3 TPM\_CertifySelfTest

### Start of informative comment:

TPM\_CertifySelfTest causes the TPM to perform a full self-test and return an authenticated value if the test passes.

If a caller itself requires proof, it is sufficient to use any signing key for which only the TPM and the caller have AuthData.

If a caller requires proof for a third party, the signing key must be one whose signature is trusted by the third party. A TPM-identity key may be suitable.

### End of informative comment.

## Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CertifySelfTest
4	4			TPM_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key that can perform digital signatures.
5	20	2S	20	TPM_NONCE	antiReplay	Anti Replay nonce to prevent replay of messages
6	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
7	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
9	20			TPM_AUTHDATA	privAuth	The authorization session digest that authorizes the inputs and use of keyHandle. HMAC key: key.usageAuth

## Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CertifySelfTest
4	4	3S	4	UINT32	sigSize	The length of the returned digital signature
5	<>	4S	<>	BYTE[]	sig	The resulting digital signature.
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: key.usageAuth



**902 Description**

903 The key in keyHandle MUST have a KEYUSAGE value of type TPM\_KEY\_SIGNING or  
904 TPM\_KEY\_LEGACY or TPM\_KEY\_IDENTITY.

905 Information returned by TPM\_CertifySelfTest MUST NOT aid identification of an individual  
906 TPM.

**907 Actions**

908 1. The TPM SHALL perform TPM\_SelfTestFull. If the test fails the TPM returns the  
909 appropriate error code.

910 2. After successful completion of the self-test the TPM then validates the authorization to  
911 use the key pointed to by keyHandle

912 a. If the key pointed to by keyHandle has a signature scheme that is not  
913 TPM\_SS\_RSASSAPKCS1v15\_SHA1, the TPM may either return TPM\_BAD\_SCHEME or  
914 may return TPM\_SUCCESS and a vendor specific signature.

915 3. Create t1 the NOT null terminated string of "Test Passed", i.e. 11 bytes.

916 4. The TPM creates m2 the message to sign by concatenating t1 || AntiReplay || ordinal.

917 5. The TPM signs the SHA-1 of m2 using the key identified by keyHandle, and returns the  
918 signature as sig.

919